ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA
INGENIERÍA DEL SOFTWARE

SMARTSNIFF

Realizado por
DANIEL ALEJANDRO CASTRO GARCÍA
Tutorizado por
ENRIQUE ALBA TORRES
JAMAL TOUTOUH EL ALAMIN
Departamento
LENGUAJE Y CIENCIAS DE LA COMPUTACIÓN

UNIVERSIDAD DE MÁLAGA
MÁLAGA, Noviembre de 2016

Fecha defensa:
El Secretario del Tribunal

Resumen: En la actualidad, nuestro entorno se ve inundado de dispositivos con interfaces de red inalámbrica. Este proyecto consiste en el desarrollo de un sistema capaz de detectar, almacenar y procesar información procedente de los dispositivos wifi y Bluetooth empleando un dispositivo móvil Android. El sistema consta de dos partes, una aplicación móvil que detectará las distintas redes inalámbricas que se encuentren a su alcance para posteriormente almacenar información sobre dichas redes, además de detectar también dispositivos Bluetooth, y una aplicación web que almacenará toda la información procedente de la aplicación móvil para posteriormente emplearla en la creación de mapas y estadísticas. El grueso del trabajo se concentra en la aplicación móvil, capaz de detectar ambos tipos de dispositivos de forma simultánea, además de la ubicación donde fueron encontrados. Los datos son almacenados de forma local en el propio dispositivo móvil hasta que el usuario los envía al servidor web, momento en el que se elaboran estadísticas que el usuario podrá consultar en la aplicación web.

Palabras claves: wifi, bluetooth, estadísticas, mapas, detección, aplicación móvil, aplicación web

Abstract: Nowadays, our environment is flooded with wireless-ready devices. This project consists in the development of a system capable of detecting, storing and processing information from wifi and Bluetooth devices using an Android mobile device. The project consists of two parts, an Android mobile application which will detect the different wireless networks within reach of the device, and a web application which stores all the information sent by the mobile application to further create maps and statistics with it. The bulk of the work is concentrated in the mobile application, which is capable of detecting both types of devices at the same time, in addition to the position where they were detected. The data is stored locally in the device until the user sends them to the server, at which point statistics and maps will be elaborated. The user will be able to see these statistics by accessing the web application.

Keywords: wifi, Bluetooth, statistics, maps, detection, mobile application, web application

Índice

Capítulo 1. Introducción

1.1. Motivación

A lo largo de los últimos años, los dispositivos inalámbricos han disfrutado de una época de auge y expansión más que notable. Los puntos de acceso wifi pueden encontrarse en cualquier lugar, desde domicilios y establecimientos privados hasta parques, calles y plazas. Asimismo, la tecnología Bluetooth se puede encontrar en prácticamente cualquier dispositivo móvil, desde teléfonos móviles hasta televisores, pasando por pulseras cuantificadoras, relojes inteligentes y coches.

Estas dos tecnologías conforman uno de los pilares fundamentales que posibilitan la existencia de las Ciudades Inteligentes (*Smart Cities*) [1]: la interconexión de dispositivos. Mediante la difusión de ambas tecnologías en el ámbito urbano, se impulsa la creación de un ecosistema digital de dispositivos interconectados. Y a pesar de que actualmente tan solo el uno por ciento de los dispositivos electrónicos está conectado a Internet [2], el aumento de esta cantidad es imparable.

La situación descrita genera grandes oportunidades en el desarrollo de aplicaciones móviles que se aprovechen de estas tecnologías. En el presente documento se explora una de ellas: se propone el desarrollo de un sistema capaz de capturar datos relevantes sobre las redes y dispositivos inalámbricos para la elaboración de mapas que muestren las redes inalámbricas desplegadas por toda un área determinada, además de diversas estadísticas para obtener información útil sobre los dispositivos y las redes de comunicación desplegadas.

1.2. Objetivos

El objetivo del presente Trabajo de Fin de Grado (TFG) es el desarrollo de una aplicación móvil que detecte redes inalámbricas (puntos de acceso) y dispositivos Bluetooth que estén en modo promiscuo. Una vez detectados, se almacenará información diversa sobre dichas redes o dispositivos, tales como el nombre y la zona geográfica donde fueron encontrados, además de diversos parámetros de configuración.

Con los datos almacenados, se procederá a elaborar estadísticas y mapas, como por ejemplo el número de redes inalámbricas wifi en un área o una clasificación de los dispositivos por fabricante.

1.3. Estructura de la memoria

Se presenta a continuación una breve descripción de los capítulos que componen este documento.

Capítulo 2: Tecnologías

Detalla las tecnologías empleadas para el desarrollo del sistema así como las herramientas que se han usado, con la correspondiente justificación de uso y comparativa con otras opciones disponibles cuando corresponda.

Capítulo 3: Especificación y análisis

Expone los requisitos del sistema y los casos de uso implementados. Se ofrece también un análisis de las bases de datos empleadas y de las interfaces de usuario, además de las limitaciones del sistema.

Capítulo 4: Diseño

Presenta los diagramas UML que han sido elaborados y usados de guía para el desarrollo del sistema. Además, se explica el diseño de las bases de datos.

Capítulo 5: Implementación y pruebas

Explica tanto la estructura de las aplicaciones del proyecto (aplicación móvil y aplicación web) como los detalles de la implementación de las partes más importantes. Posteriormente se detalla la parte de pruebas.

Capítulo 6: Conclusiones y líneas futuras

Describe las conclusiones a las que se han llegado como consecuencia del desarrollo del sistema, además de plantear una serie de líneas futuras.

Capítulo 2. Tecnologías

En esta sección se van a detallar las tecnologías y herramientas empleadas durante el desarrollo del sistema. Se presentan algunas alternativas encontradas durante el proceso de desarrollo y se justifica la decisión tomada.

2.1. Repositorio de código fuente

Se ha empleado Git como software de control de versiones para la totalidad del código del proyecto, por facilidad de uso y rendimiento. El código del proyecto está alojado en GitHub, plataforma que utiliza el sistema de control de versiones antes mencionado. Los enlaces a los repositorios son los siguientes:

- Aplicación móvil: https://github.com/dandev237/SmartSniff
- Aplicación web: https://github.com/dandev237/smartsniff-webserver

2.2. Aplicación móvil

La aplicación móvil se ha implementado para la plataforma Android, puesto que es el sistema operativo más extendido en dispositivos móviles. Así, todas las tecnologías empleadas para el desarrollo de la aplicación móvil están ligadas a esta decisión.

2.2.1. Base de datos

En la actualidad podemos encontrar dos sistemas gestores de bases de datos para aplicaciones móviles principales, SQLite [3] y Realm. Al ser SQLite la más extendida, probada y documentada, se ha optado por esta frente a Realm.

2.2.2. Bibliotecas de funciones específicas para Android

Los principales módulos empleados en la aplicación Android y la funcionalidad a la que dan soporte se exponen en la Tabla 1.

Nombre	Funcionalidad
Google Play Services [4]	API de Google para Android, que contiene extensa funcionalidad de diversos tipos. En nuestra aplicación, se emplea este módulo para dar soporte a la funcionalidad de geolocalización por GPS.
Google Maps Android API [5]	API de Google para Android con funcionalidad enfocada a mapas. En nuestra aplicación, se emplea este módulo para mostrar un mapa de calor en la interfaz principal.
Volley [6]	Biblioteca HTTP para Android que facilita y acelera las tareas de comunicación HTTP con respecto al framework Android estándar. En nuestra aplicación, se emplea este módulo para implementar funcionalidad de comunicación entre cliente y servidor mediante peticiones HTTP.
Gson (Google Gson) [7]	Biblioteca Java que permite la serialización y deserialización entre objetos Java y su representación en formato JSON. En nuestra aplicación, se emplea este módulo para implementar funcionalidad de creación de objetos JSON empleados para enviar datos al servidor web mediante las peticiones HTTP manejadas con Volley.

Tabla 1 – Tecnología usada en la aplicación Android

2.3. Servidor web

A continuación se describe la base de datos y las herramientas principales para el desarrollo de la aplicación web.

2.3.1. Base de datos

La base de datos contenida en el servidor web se ha desarrollado empleando PostgreSQL [8], un sistema de gestión de bases de datos relacional orientado a objetos publicado bajo una licencia de software libre.

Se ha escogido este sistema debido a su amplia variedad de tipos nativos. Entre ellos, se encuentra el tipo "*point*", empleado de forma extensa en proyectos que emplean datos geolocalizados, como lo es este. Disponer de este tipo nativo ha sido de gran utilidad y ha facilitado mucho el desarrollo del sistema.

Inicialmente, se planteó la posibilidad de emplear el módulo PostGIS junto a PostgreSQL debido a la enorme cantidad de funcionalidad que aporta en cuanto a geolocalización. No obstante, al comprobar que el tipo nativo "*point*" satisfacía los requisitos de funcionalidad, se optó por descartar dicho módulo.

2.3.2. Servidor HTTP

Para desarrollar el servidor web se ha empleado Kestrel, un servidor web multiplataforma basado en una librería asíncrona también multiplataforma.

Kestrel está diseñado para ser ejecutado detrás de un *proxy*, por lo que esta cuestión planteó una elección a tomar en cuanto a qué servidor *proxy* inverso emplear. Un servidor *proxy* inverso nos permite descargar trabajo del servidor web como servir contenido estático, cachear y comprimir peticiones [9]. El servidor *proxy* inverso puede ser ejecutado tanto en una máquina dedicada como en la misma máquina en la que se encuentra desplegado el servidor web. En el caso de este sistema, por cuestiones de desarrollo se ha optado por la segunda opción.

Al ser nuestro entorno de producción un sistema Linux, las opciones originales eran Apache o Nginx. Debido a la facilidad de uso y a la gran cantidad de documentación disponible para desarrollo de aplicaciones web con el framework .NET Core y Nginx para entornos Linux, se ha optado por este último como servidor *proxy* inverso para este sistema.

2.4. Aplicación web

Todos los módulos empleados para el desarrollo de la aplicación web y la funcionalidad a la que dan soporte se exponen en la Tabla 2.

2.5. Entorno de desarrollo

Para desarrollar el sistema se han empleado dos entornos de desarrollo:

- Para la aplicación móvil, se ha empleado Android Studio en su versión 2.1.2, junto con las herramientas que integra (como por ejemplo Gradle).
- Para la aplicación web, se ha empleado Visual Studio 2015.

Usamos estos dos IDEs debido a su rendimiento y a que son los entornos de desarrollo *de facto* para las aplicaciones que se han desarrollado en este TFG.

Se ha empleado la herramienta software SINVAS UML para la elaboración de diagramas de modelado, por su facilidad de uso y por ser gratuita en su versión *Community Edition*. Para la gestión de la base de datos del servidor se ha empleado el programa pgAdmin por la amplia gama de funcionalidad que ofrece.

Nombre	Funcionalidad
.NET Core [10]	Framework multiplataforma y de código abierto de Microsoft que consiste en una versión modular de .NET Framework. Es empleado para desarrollar diversos tipos de aplicaciones, como aplicaciones de escritorio y Windows Phone. En nuestra aplicación, .NET Core constituye la base sobre la que se ha desarrollado la aplicación web.
JQuery	Biblioteca JavaScript creada para simplificar la forma de manejar documentos HTML mediante JavaScript y añadir interacción con AJAX a páginas web. En nuestra aplicación, se emplea para cargar la información de la base de datos recibida mediante objetos JSON en el mapa de calor y las estadísticas de la aplicación web.
Leaflet [11]	Biblioteca JavaScript de código abierto usada para crear aplicaciones web que contengan mapas. En nuestra aplicación, se emplea para implementar la funcionalidad que crea y muestra un mapa en la aplicación web.
Leaflet.heat js	Plugin sencillo para Leaflet que añade funcionalidad de mapas de calor. En nuestra aplicación, se emplea para mostrar información en forma de mapa de calor sobre el mapa implementado con Leaflet.
Heatmap.js [12]	Plugin para Leaflet que funciona sobre el plugin leaflet.heat js, con el objetivo de hacer su uso más sencillo para el desarrollador y añadir nueva funcionalidad. En nuestra aplicación, se usa este plugin en conjunción con Leaflet.heat js para implementar el mapa de calor.
FusionCharts [13]	Biblioteca JavaScript para dibujar estadísticas en forma de gráficas. En nuestra aplicación, se emplea para mostrar las estadísticas elaboradas por la aplicación web de forma gráfica e intuitiva para el usuario.

Tabla 2 – Tecnología usada en la aplicación web

Capítulo 3. Especificación y análisis

Este capítulo expone los requisitos del sistema y los casos de uso implementados, además de un análisis de las bases de datos empleadas y de las interfaces de usuario. Concluye con las limitaciones del sistema.

3.1. Requisitos de la aplicación Android

A continuación se describen los actores identificados para la aplicación Android y los requisitos de la misma.

3.1.1. Actores

Debido a la propia composición del problema, únicamente consideramos un actor principal, el Usuario. Todos los casos de uso de la aplicación han sido modelados considerando al Usuario como actor principal. No es necesario considerar un perfil de Administrador puesto que la aplicación no emplea datos personales que constituyan perfil de usuario alguno que se deba gestionar.

Adicionalmente se identifica un actor secundario, el Servidor. Dicho actor representa la máquina donde la aplicación web se está ejecutando, y se encargará de recibir los datos enviados por el Usuario mediante la aplicación móvil, tal y como se indica más adelante.

3.1.2. Requisitos funcionales

En la Tabla 3 pueden encontrarse los requisitos funcionales, los cuales definen la funcionalidad de la aplicación y las acciones que puede llevar a cabo el Usuario o la aplicación en respuesta a alguna acción de este.

Identificador	Descripción
RF-1	El usuario podrá iniciar un escaneo para detectar puntos de acceso wifi y dispositivos Bluetooth al alcance del dispositivo móvil.
RF-2	El usuario podrá detener un escaneo en progreso.
RF-3	La aplicación mostrará los resultados de un escaneo que haya finalizado.
RF-4	La aplicación almacenará de forma local los resultados de los escaneos.
RF-5	El usuario podrá eliminar en cualquier momento los datos almacenados de forma local.
RF-6	El usuario podrá enviar toda la información obtenida de los escaneos a un sistema de almacenamiento remoto.
RF-7	La aplicación mostrará un mapa de calor, que permitirá al usuario visualizar de forma gráfica los datos recogidos.
RF-8	El usuario podrá configurar determinados parámetros de la aplicación.

Tabla 3 – Requisitos funcionales de la aplicación Android

3.1.3. Requisitos no funcionales

Los requisitos no funcionales determinan condiciones o restricciones sobre las que el sistema funciona o es desarrollado. Se presentan en la Tabla 4 clasificados en función de la categoría correspondiente según el estándar IEEE-Std 830-1993.

Identificador	Categoría	Descripción
RNF-1	Rendimiento	La aplicación debe procesar un escaneo de dispositivos wifi y Bluetooth con un impacto mínimo en el rendimiento.
RNF-2	Fiabilidad	La aplicación debe ser capaz de mantener un escaneo en proceso en segundo plano.
RNF-3	Fiabilidad	La aplicación debe ser capaz de mantener su funcionamiento ante posibles deshabilitaciones de las interfaces empleadas durante los escaneos (wifi, Bluetooth, GPS) del dispositivo móvil.
RNF-4	Fiabilidad	La aplicación deberá reaccionar ante posibles errores en la manipulación de la base de datos local de forma que no interrumpa su funcionamiento.
RNF-5	Interoperabilidad	La aplicación debe poder enviar los datos almacenados localmente al servidor mediante una API REST.
RNF-6	Documentación	La aplicación estará bien documentada en un manual de usuario.
RNF-7	Mantenimiento	Los fragmentos de código de la aplicación más críticos y/o empleados durante la ejecución deberán estar debidamente documentados.
RNF-8	Almacenamiento	Los datos obtenidos por la aplicación deberán ser almacenados en una base de datos local.
RNF-9	Usabilidad	La aplicación tendrá una interfaz que haga que su uso sea sencillo para cualquier usuario.
RNF-10	Interfaz	La interfaz de la aplicación incluirá únicamente los elementos mínimos necesarios para ser empleada con efectividad.

Tabla 4 – Requisitos no funcionales de la aplicación Android

3.2. Casos de uso de la aplicación Android

La Figura 1 presenta todos los casos de uso necesarios para cubrir todos los requisitos funcionales de la aplicación.

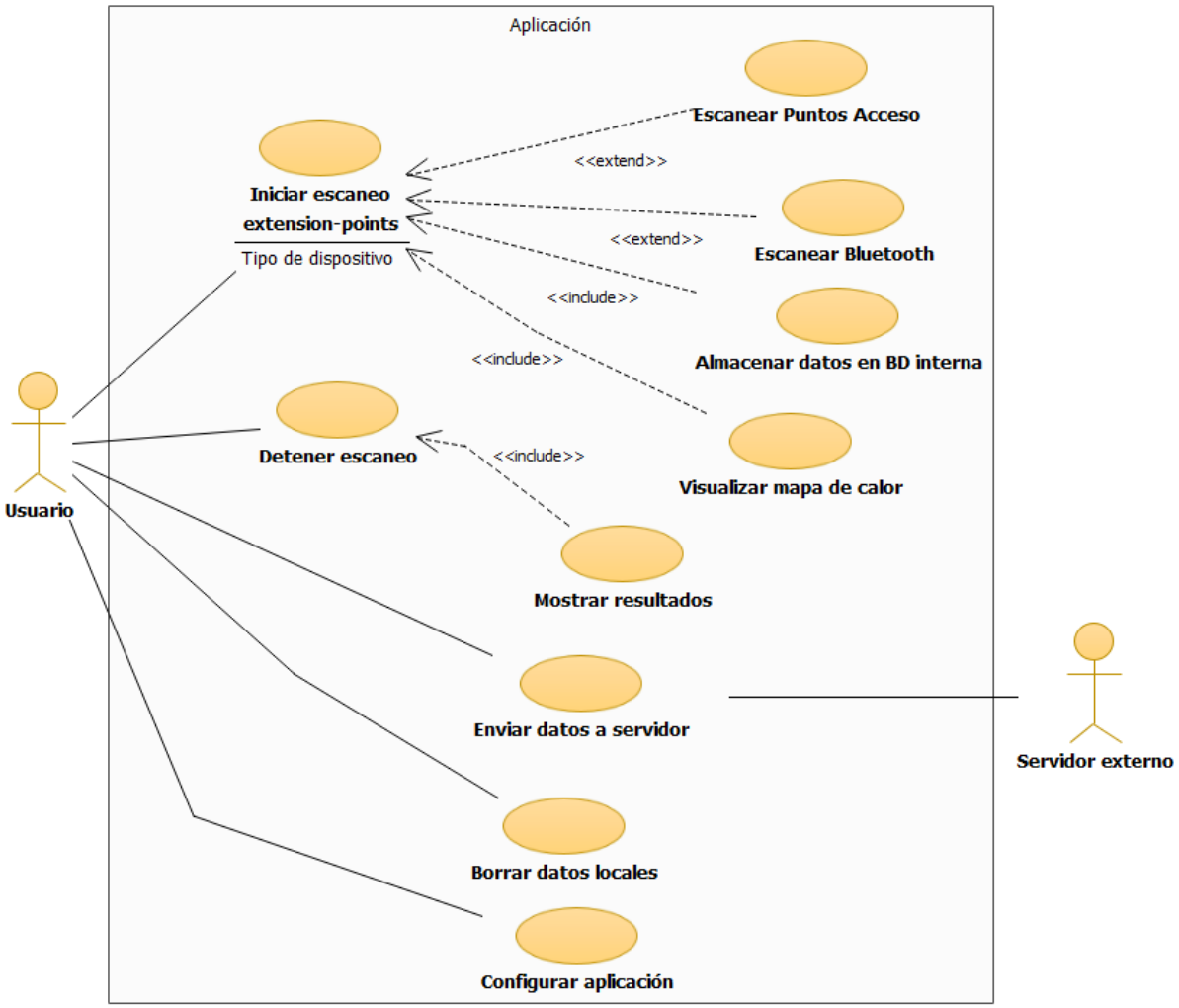

Figura 1 – Diagrama de casos de uso de la aplicación Android

Las tablas 5, 6, 7, 8 y 9 contienen la especificación de los cinco casos de uso principales (aquellos conectados directamente con el Usuario del diagrama del apartado anterior): Iniciar escaneo, Detener escaneo, Enviar datos a servidor, Borrar datos locales y Configurar aplicación.

Nombre	Iniciar escaneo
Identificador	UC-01
Requisitos funcionales asociados	RF-1, RF-4, RF-7
Descripción	Este caso de uso describe la funcionalidad que permite al usuario iniciar un escaneo en busca de dispositivos wifi y Bluetooth al alcance de su dispositivo móvil
Precondiciones	La aplicación está en *standby* (sin escaneo activo)
Postcondiciones	La aplicación está en modo escaneo
Prioridad	Alta
Escenario principal	1. El usuario presiona el botón de iniciar escaneo 2. La aplicación pide permiso para activar las interfaces del dispositivo necesarias (GPS, Bluetooth) 3. La aplicación inicia los procesos de detección con las interfaces disponibles 4. A medida que la aplicación encuentra datos, los mostrará en el mapa de calor de la interfaz
Escenario alternativo	Ninguno

Tabla 5 – Caso de uso 1: Iniciar escaneo

Nombre	Detener escaneo
Identificador	UC-02
Requisitos funcionales asociados	RF-2, RF-3
Descripción	Este caso de uso describe la funcionalidad que permite al usuario interrumpir un escaneo en progreso
Precondiciones	La aplicación está en modo escaneo
Postcondiciones	La aplicación está en *standby* (sin escaneo activo)
Prioridad	Alta
Escenario principal	1. El usuario presiona el botón para cancelar el escaneo activo 2. La aplicación detiene los procesos de detección 3. La aplicación muestra una interfaz en la que se detallan los resultados del escaneo detenido 4. La aplicación actualiza el mapa de calor de la interfaz
Escenario alternativo	Ninguno

Tabla 6 – Caso de uso 2: Detener escaneo

Nombre	Borrar datos locales
Identificador	UC-03
Requisitos funcionales asociados	RF-5
Descripción	Este caso de uso describe la funcionalidad que permite al usuario eliminar de la base de datos local los resultados de los escaneos almacenados en ella
Precondiciones	La aplicación está en *standby* (sin escaneo activo)
Postcondiciones	Los datos de la base de datos local son eliminados
Prioridad	Media
Escenario principal	1. El usuario presiona el botón de borrar datos 2. La aplicación pide una confirmación al usuario 3. El usuario acepta la confirmación 4. La aplicación elimina todos los datos almacenados en la base de datos local 5. La aplicación muestra una notificación al usuario para indicar que el borrado se ha realizado con éxito
Escenarios alternativos	El usuario deniega la confirmación pedida por la aplicación: 3.a – El usuario deniega la confirmación 4.a – La aplicación cancela el proceso de borrado No hay datos que borrar: 2.b - La aplicación muestra un mensaje, indicando al usuario que no hay datos almacenados.

Tabla 7 – Caso de uso 3: Borrar datos locales

Nombre	Enviar datos a servidor
Identificador	UC-04
Requisitos funcionales asociados	RF-6
Descripción	Este caso de uso describe la funcionalidad que permite al usuario enviar los resultados de los diversos escaneos realizados a un servidor para su almacenamiento
Precondiciones	La aplicación está en *standby* (sin escaneo activo)
Postcondiciones	Los datos disponibles se han enviado al servidor
Prioridad	Media
Escenario principal	1. El usuario presiona el botón de envío de datos 2. La aplicación recolecta los datos almacenados de forma local y los envía al servidor
Escenario alternativo	No hay datos para enviar: 2.b - La aplicación muestra un mensaje, indicando al usuario que no hay datos que enviar.

Tabla 8 – Caso de uso 4: Enviar datos a servidor

Nombre	Configurar aplicación
Identificador	UC-05
Requisitos funcionales asociados	RF-8
Descripción	Este caso de uso describe la funcionalidad que permite al usuario configurar la aplicación mediante un menú habilitado a tal efecto
Precondiciones	La aplicación está en *standby* (sin escaneo activo)
Postcondiciones	La aplicación queda configurada con los parámetros de configuración introducidos por el usuario
Prioridad	Alta
Escenario principal	1. El usuario presiona el botón para abrir el menú de configuración 2. La aplicación muestra un menú de configuración 3. El usuario altera los parámetros que considere oportunos, y cierra el menú
Escenario alternativo	Ninguno

Tabla 9 – Caso de uso 5: Configurar aplicación

3.3. Análisis de las bases de datos

A continuación se presentan las especificaciones de las bases de datos del proyecto. En la Sección 4.2 se presenta el modelo relacional con mayor cantidad de detalles.

3.3.1. Aplicación Android

La base de datos Android contiene tres tipos de entidades:

- **Session:** almacena información de las sesiones. Una sesión contiene la fecha de inicio de un escaneo y su fecha de finalización.
- **Device:** contiene información sobre un dispositivo detectado, sea wifi o Bluetooth. Incluye, entre otros parámetros, el SSID, la dirección MAC y el fabricante del dispositivo, si es posible encontrarlo.
- **Location:** almacena información sobre las localizaciones en las que se han encontrado uno o varios dispositivos. Se almacena la fecha a la que se registró la localización y las coordenadas de la misma.

Además de estas tres entidades, hay una más, **Association**. Esta entidad guarda información de las asociaciones entre las tres entidades anteriores en la base de datos.

3.3.2. Servidor

La base de datos del servidor es exactamente igual que la base de datos de la aplicación Android con la diferencia de que la entidad **Session** almacena, además de los datos anteriores, la dirección MAC del adaptador de red del dispositivo móvil que envía los datos.

De esta manera, se facilitan las comprobaciones de unicidad de las sesiones en el servidor y se provee de una manera básica de identificación de las sesiones por usuario, preparando así el terreno para una posible implementación de estadísticas personales en la propia aplicación móvil.

3.4. Análisis de la interfaz de usuario

La interfaz de usuario se ha desarrollado considerando en todo momento la facilidad de manejo de la aplicación. Es por ello que se ha diseñado y construido la interfaz minimizando el número de pulsaciones sobre la pantalla que el usuario debe realizar.

Se han considerado los siguientes puntos a la hora de diseñar la interfaz:

- **Minimalismo:** minimización de los elementos presentes en la interfaz de usuario.
- **Navegación:** en la aplicación móvil el usuario tiene a su disposición un menú en la parte superior derecha de la aplicación móvil que le permitirá acceder a las funciones de enviar y borrar datos, además de al menú de configuración. En la aplicación web, el menú se localiza también en la parte superior derecha, y permite al usuario acceder al mapa de calor de dispositivos detectados, a la página que contiene las estadísticas y a una página que introduce el sistema a los usuarios.
- **Aspecto visual:** se ha procurado combinar el minimalismo de interfaz comentado anteriormente con un aspecto visual sencillo para el usuario en las dos aplicaciones del sistema.

Capítulo 4. Diseño

Este capítulo presenta los diagramas UML que han sido creados y usados de guía para el desarrollo del sistema. Se detalla además el modelo de base de datos elaborado para las dos bases de datos.

4.1. Diagrama de distribución

La comunicación es unidireccional en el caso de la aplicación móvil y bidireccional en el caso de la aplicación web. El servidor web también se comunica de manera bidireccional con la base de datos del propio servidor. De igual manera, la aplicación Android se comunica bidireccionalmente con la base de datos interna del dispositivo. La Figura 2 ilustra estos conceptos.

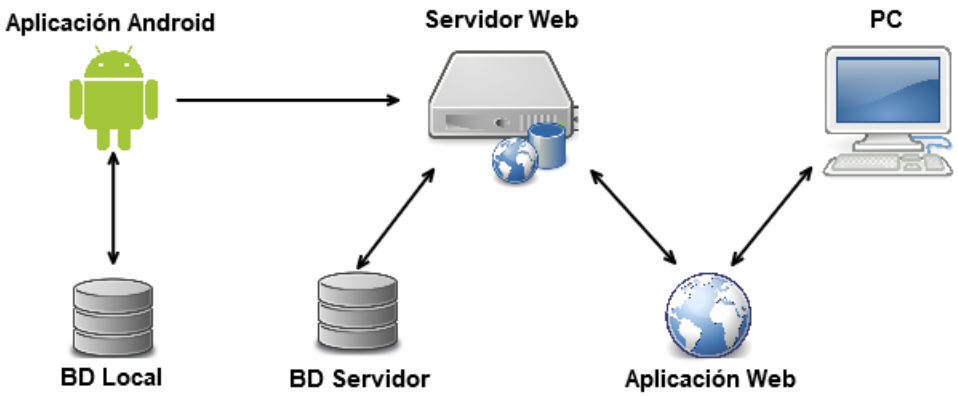

Figura 2 – Diagrama de distribución

El usuario puede hacer uso del sistema de dos maneras bien diferenciadas:

1. **Usando la aplicación móvil**: el usuario realiza escaneos con la aplicación móvil para recabar información sobre los dispositivos inalámbricos de su entorno, que después enviará al servidor web mediante una petición HTTP de tipo POST que incluye un objeto JSON con la información de la base de datos local.

2. **Usando la aplicación web**: el usuario puede consultar en cualquier momento el mapa de calor de dispositivos encontrados y las estadísticas elaboradas a partir de los datos enviados por la aplicación móvil. Además, puede consultar una breve introducción al sistema.

4.2. Modelo relacional de las bases de datos

A continuación se detalla el modelo entidad-relación de las dos bases de datos del proyecto, entrando en detalle sobre cuestiones de diseño.

4.2.1. Local

El modelo de la Figura 3 representa las tablas existentes en la base de datos de la aplicación móvil junto a sus relaciones. No obstante, está hecho en MySQL Workbench puesto que es la única herramienta encontrada capaz de generar estas representaciones, muy útiles como esquema para el desarrollo de la base de datos real.

Hay que considerar que la imagen del modelo como tal no es representativa ya que la base de datos local está hecha en SQLite. Los tipos de datos empleados en la base de datos local se comentan a continuación.

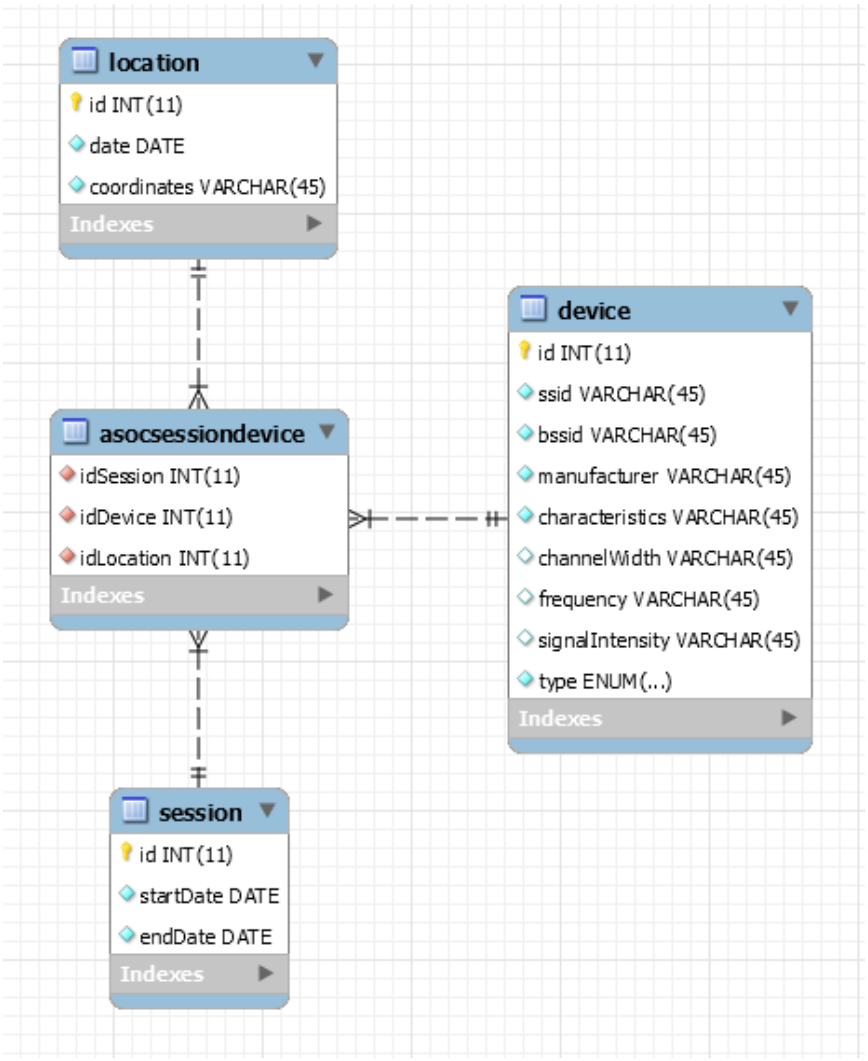

Figura 3 – Modelo de base de datos local

Las tablas que aparecen en el modelo y sus columnas se encuentran explicadas a continuación:

- **Tabla session:** en esta tabla se almacena información de las sesiones de escaneo que el usuario ha llevado a cabo. Una sesión es un escaneo completo.
    - **startDate (tipo TEXT):** fecha a la que comenzó el escaneo.
    - **endDate (tipo TEXT):** fecha a la que terminó el escaneo.
- **Tabla location:** en esta tabla se almacena información de las localizaciones en las que se han encontrado dispositivos inalámbricos.
    - **date (tipo TEXT)**: fecha a la que se encontró el dispositivo.
    - **coordinates (tipo TEXT):** coordenadas geográficas de la ubicación.
- **Tabla device:** en esta tabla se almacena información de los dispositivos inalámbricos detectados. En la misma tabla se almacena información tanto de dispositivos wifi como Bluetooth.
    - **ssid (tipo TEXT):** nombre del dispositivo inalámbrico. En el caso de dispositivos Bluetooth, es el nombre que el usuario asigna al dispositivo (*friendly name*).
    - **bssid (tipo TEXT):** dirección física (*MAC Address)* del adaptador adecuado del dispositivo detectado (si es un punto de acceso wifi será la del adaptador de red, si es un dispositivo Bluetooth será la del adaptador Bluetooth). Este campo es único.
    - **manufacturer (tipo TEXT):** fabricante del dispositivo.
    - **characteristics (tipo TEXT):** este campo almacena información diferente en función del tipo de dispositivo:
        - **wifi:** almacena los esquemas de autenticación, manejo de claves y encriptación a los cuales da soporte el punto de acceso.
        - **Bluetooth:** almacena el tipo de dispositivo principal del dispositivo Bluetooth detectado (AUDIO_VIDEO, COMPUTER, HEALTH, IMAGING, MISC, NETWORKING, PEROPHERAL, PHONE, TOY, WEARABLE).
    - Los siguientes tres campos se emplean únicamente si el dispositivo es un punto de acceso wifi:
        - **channelWidth (tipo TEXT):** ancho de banda del canal.
        - **frequency (tipo INTEGER):** frequencia de emisión. A partir de este dato se puede extraer la banda de frequencia en la que emite el dispositivo.
        - **signalIntensity (tipo INTEGER):** potencia de la señal detectada en dBm (decibelios-milivatios).
    - **type (tipo TEXT)**: tipo del dispositivo (WIFI, BLUETOOTH)

- **Tabla asocsessiondevice**: en esta tabla se guardan las asociaciones entre sesión, ubicación y dispositivo. La asociación podría mencionarse con la siguiente frase: “En la sesión x, en la ubicación y, se encontró el dispositivo z”. Los tres campos de la tabla son claves foráneas, y la clave primaria es una composición de las tres.
    - **idSession (tipo INTEGER):** identificador de la sesión.
    - **idDevice (tipo INTEGER):** identificador del dispositivo.
    - **idLocation (tipo INTEGER):** identificador de la ubicación.

4.2.2. Remota

El modelo de la base de datos del servidor es exactamente igual que el de la base de datos de la aplicación móvil, con el añadido del campo **macAddress** en la tabla **Session** que almacena la dirección MAC del adaptador de red del dispositivo móvil que ha enviado la sesión, además del resto de datos almacenados por su aplicación. Se ha introducido este cambio para identificar las sesiones en función del dispositivo que las envía, como se comentó en la sección 3.3.2.

La base de datos del servidor está hecha en PostgreSQL, y presenta un cambio significativo con respecto a la de la aplicación móvil en cuanto a tipado de datos: los tipos de las columnas cambian a uno más apropiado para el dato que almacenan.

4.3. Diagrama de clases

La Figura 4 es el diagrama de clases UML de la aplicación Android. El diagrama está simplificado, pero si se desea consultar una versión completa del mismo se puede encontrar en el Anexo técnico entregado junto a esta memoria.

La aplicación web está compuesta por las siguientes seis clases:

- Cuatro clases de modelo, equivalentes a las de la aplicación móvil (Device, Location, Session, AsocSessionDevice).
- Dos clases que cumplen la función de controlador en el patrón MVC (Modelo Vista Controlador) propio de .NET:
    - **HomeController**: clase controladora dedicada a manejar la funcionalidad de las páginas web a nivel de controlador.
    - **DbController**: clase controladora encargada de gestionar todas las transacciones con la base de datos del servidor, tanto de creación como de obtención de datos. Contiene métodos RESTful.

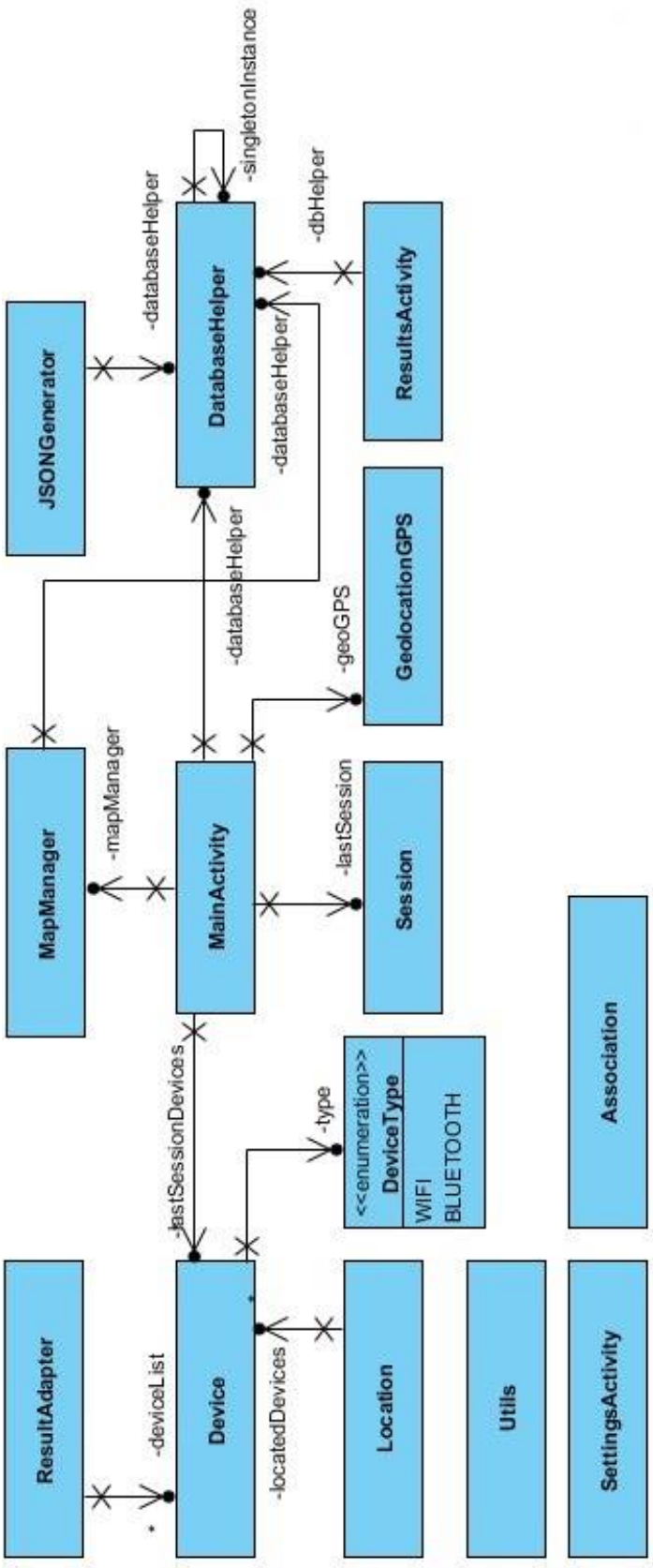

Figura 4 – Diagrama de clases de la aplicación Android

Capítulo 5. Implementación y pruebas

Este capítulo explica tanto la estructura de las aplicaciones del TFG como los detalles de la implementación de las partes más importantes. Concluye con una presentación de las pruebas realizadas.

5.1. Estructura del proyecto Android

La Figura 5 muestra la estructura del proyecto de la aplicación Android, haciendo énfasis en las clases. La aplicación está contenida en el paquete xyz.smartsniff.

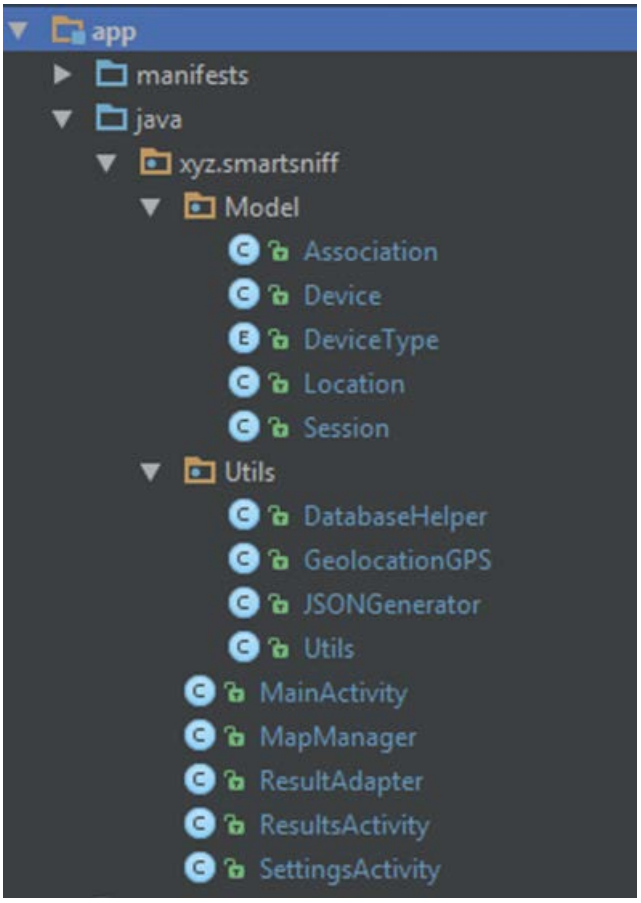

Figura 5 – Estructura del proyecto de la aplicación Android

A continuación se detallan las clases del proyecto, las cuales aparecen en la figura anterior:

- **Model:** paquete que contiene las clases que conforman el modelo de datos de la aplicación. Se compone de cuatro clases y un enumerado.
    - **Association, Device, Location, Session:** las cuatro entidades que se almacenan en la base de datos y con las que trabaja la aplicación.

    - **DeviceType:** enumerado que contiene los valores que los objetos de tipo Device pueden almacenar en su atributo **type**: WIFI o BLUETOOTH.
- **Utils:** paquete que contiene clases que realizan tareas de cómputo de forma transparente al usuario para el adecuado funcionamiento de la aplicación.
    - **DatabaseHelper:** clase que implementa la base de datos local en SQLite, además de los métodos pertinentes para añadir, obtener y eliminar datos de sus tablas.
    - **GeolocationGPS:** clase que, empleando el módulo **Google Play Services API**, realiza todas las tareas de geolocalización requeridas por la aplicación.
    - **JSONGenerator:** clase dedica a generar los objetos JSON que la aplicación usa para enviar toda la información contenida en su base de datos local. Para ello, se apoya de dos módulos: **Gson** y **Volley**. El primero se usa para serializar los objetos de las clases modelo en el JSON y el segundo se usa para crear el JSON y enviarlo en una petición HTTP de tipo POST.
    - **Utils:** clase genérica con métodos, constantes y atributos que no tenían lugar en una clase concreta de la aplicación y que, por lo general, son usados en varias partes del código.
- En el paquete principal de la aplicación del proyecto nos encontramos con clases que contienen la lógica que opera por debajo de las vistas de la aplicación, es decir, las clases controladoras.
    - **MainActivity:** actividad principal de la aplicación. Es donde se desarrolla la actividad de escaneo, donde se muestra el mapa de calor y donde aparece el menú que contiene el resto de funcionalidades de la aplicación.
    - **MapManager:** clase dedicada a gestionar el mapa de calor de la actividad principal.
    - **ResultAdapter:** adaptador personalizado de la *ListView* que gestiona los resultados de los escaneos que aparecen al terminar estos.
    - **ResultsActivity:** actividad que se activa cada vez que el usuario termina un escaneo. Muestra los resultados de dicho escaneo con todo lujo de detalles: fecha de inicio, fecha de fin, número total de hallazgos y una lista con todos los hallazgos. Gracias a la funcionalidad implementada en **ResultAdapter**, el usuario puede pulsar en un elemento de la lista para que aparezcan datos adicionales sobre dicho elemento.
    - **SettingsActivity:** actividad que se activa cuando el usuario pulsa la opción de configuración del menú de la aplicación. Contiene los parámetros de configuración que el usuario puede personalizar, como el modo de ahorro de energía.

5.2. Estructura del proyecto web

En esta sección se procederá de igual manera que en la sección anterior, pero con la estructura del proyecto web. La Figura 6 expone dicha estructura y, seguidamente, se encuentra una explicación de las partes marcadas en la misma.

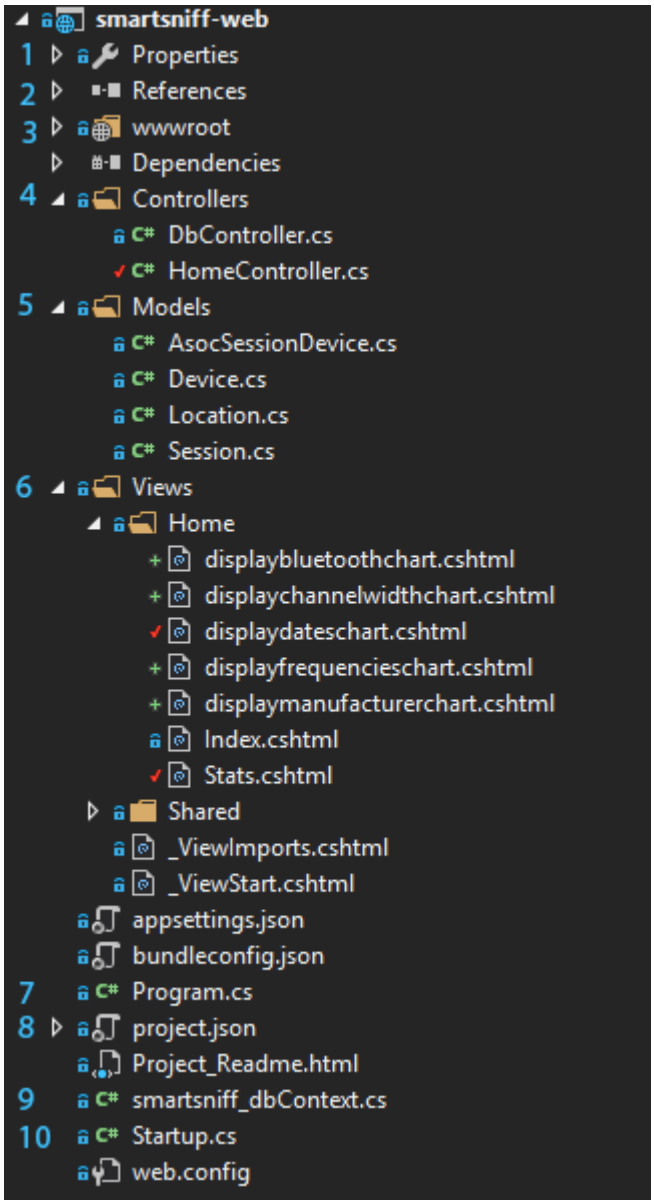

Figura 6 – Estructura del proyecto de la aplicación web

Las partes importantes de la estructura se encuentran numeradas en la Figura 6. Están detalladas a continuación:

1. **Properties:** aquí es donde se almacena la configuración general y las propiedades del proyecto.
2. **References:** apartado en el que se guardan todas las referencias a librerías de terceros o externas.

3. **wwwroot:** en esta carpeta están contenidos todos los ficheros JavaScript, CSS, imágenes, etc. Es la carpeta raíz de la página web.
4. **Controllers:** en esta carpeta están contenidas todas las clases controladoras de la aplicación web.
    a. **DbController:** clase encargada de gestionar todas las transacciones con la base de datos del servidor.
    b. **HomeController:** clase encargada de manejar la funcionalidad de las páginas web a nivel de controlador.
5. **Models:** en esta carpeta están contenidas las clases que conforman el modelo de datos de la aplicación. Son las mismas 4 clases de entidad que aparecen en la aplicación móvil, con la diferencia de que la clase **AsocSessionDevice** se corresponde con la clase **Association** de la aplicación móvil.
6. **Views:** en esta carpeta están contenidas todas las vistas de la aplicación web, es decir, la estructura de las páginas en lenguaje HTML.
7. **Program.cs:** clase que conforma el punto de partida de la aplicación web. Es en esta clase donde se construye y ejecuta el servidor web autocontenido por la propia aplicación, que será el encargado de escuchar las peticiones HTTP.
8. **project.json:** fichero en formato JSON que contiene la configuración global del proyecto.
9. **smartsniff_dbContext.cs:** clase que es empleada para consultar la base de datos y agrupar los cambios que puedan realizarse en los datos para escribirlos en la propia base de datos como una unidad.
10. **Startup.cs:** antes de que la aplicación se ejecute, esta clase es la encargada de realizar un proceso de configuración que aglutina las librerías a emplear y la inyección de dependencias, entre otros pasos.

5.3. Desarrollo del proyecto

Esta sección explica el desarrollo de las actividades más críticas del sistema, empezando por la aplicación móvil y terminando por la aplicación web. Las tareas se presentan siguiendo un orden cronológico, reflejando una secuencia habitual de ejecución de la aplicación, pasando por toda su funcionalidad. Son las siguientes:

- Aplicación móvil
    - Branded Splash Screen
    - Escanear dispositivos
    - Detener escaneo
    - Enviar datos
    - Borrar datos
    - Configuración
- Aplicación web
    - Mapa de calor
    - Estadísticas

5.3.1. Branded Splash Screen

En las primeras fases de desarrollo de la aplicación móvil se podía apreciar que cada vez que se ejecutaba la aplicación, esta mostraba una pantalla completamente blanca y vacía durante unos segundos en los cuales estaba cargando la interfaz principal (el *layout* de la interfaz y el mapa de calor).

Para solventar este inconveniente se pensó en desarrollar lo que se conoce como una *splash screen.* Esto es una pantalla que aparece nada más abrir la aplicación y que muestra una imagen representando el logo de la misma, y que tiene como propósito ocultar la interfaz del programa al usuario mientras este se carga. Están recogidas en la guía de Google de *Material Design* para su interfaz de Android [14]. Para implementar esta solución se ha procedido como sigue.

En primer lugar, se debe declarar un tema visual para la pantalla que aparecerá. Esto se hace en el fichero styles.xml contenido en la carpeta res/values del proyecto.

```
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="AppTheme.BrandedLaunch" parent="AppTheme">
        <item
name="android:windowBackground">@drawable/branded_logo</item>
    </style>
```

Código 1 – Tema visual de la *splash screen*

Como se puede observar, el tema visual principal es padre del tema visual empleado en la pantalla. Esto se hace para que la transición entre la *splash screen* y la interfaz principal sea lo más transparente posible para el usuario.

Lo siguiente que debemos hacer es construir el fichero que en el tema visual hemos colocado bajo la etiqueta **android:windowBackground**. Esto es, el elemento que contendrá los elementos visuales de la pantalla. El fichero en cuestión se denomina branded_logo.xml y está en la carpeta res/drawable del proyecto, y contiene la estructura de la pantalla inicial:

- Un elemento que contiene un color de fondo blanco, acorde al tema visual de la aplicación.
- Un elemento que contiene la imagen que se muestra en la pantalla de carga. En nuestro caso, es el icono de la aplicación pero a una resolución mayor, y en formato de imagen *9-patch* para que Android pueda mostrarla a un tamaño adecuado a la pantalla del usuario, sea cual sea su resolución.

```
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">
    <item>
        <color android:color="@color/background_material_light"/>
    </item>
    <item>
        <bitmap
        android:src="@drawable/smartsniff_launch9patch"
        android:tileMode="disabled"
        android:gravity="fill_horizontal|fill_vertical"/>
    </item>
</layer-list>
```

Código 2 – Elemento visual de la *splash screen*

Por último, asignamos el tema visual que acabamos de crear a la actividad principal de la aplicación en el manifiesto de la misma (fichero AndroidManifest.xml). Cabe destacar que, para que la aplicación recupere su tema principal, hay que hacer una llamada a la función **setContentView()** con la interfaz principal de la aplicación al principio del método **onCreate()** de la actividad principal.

El resultado final es el mostrado en la Figura 7. Ahora, cada vez que el usuario ejecute la aplicación, será esta pantalla la que salga en primer lugar, evitando así mostrar una pantalla vacía durante los segundos en los que la aplicación se carga.

Además, al estar la interfaz contenida en un fichero XML, está precargada en memoria, por lo que la pantalla se muestra inmediatamente sin tiempo de carga alguno.

Figura 7 – *Branded Splash Screen*

5.3.2. Escanear dispositivos

Como seguramente intuirá el lector, esta funcionalidad es crítica para el funcionamiento de la aplicación. De hecho, es en torno a la cual gira el resto del sistema, desde la aplicación móvil hasta la aplicación web. Un esquema de funcionamiento a alto nivel es el siguiente:

1. El usuario presiona el botón de escaneo.
2. La aplicación crea una nueva sesión, y toma la fecha actual como fecha de inicio.

3. La aplicación verifica que los permisos de ubicación y la interfaz Bluetooth del dispositivo están disponibles. En caso contrario, pide al usuario permiso para activar las que no lo estén.
4. El sistema activa los procedimientos de escaneo, por lo que se empieza a escanear tanto puntos de acceso wifi como dispositivos Bluetooth de forma simultánea.
5. Cuando se detecta un dispositivo:
    a. La aplicación pide la ubicación del dispositivo móvil, proveniente o bien del GPS o bien de Internet, en función de si el usuario tiene el modo ahorro de energía activado o no. Dicha ubicación se almacena en una variable temporal.
    b. Una vez hecho esto, se crea un objeto de tipo Device y se puebla con la información obtenida del escaneo.
    c. La aplicación verifica si el dispositivo está ya en la base de datos o no, es decir, si ha sido descubierto previamente:
        i. Si no está en la base de datos, se inserta a la base de datos y se realiza una asociación entre la sesión, el dispositivo y la ubicación. Además, se incrementa el número de hallazgos de la sesión.
        ii. Si está en la base de datos, no se realiza la inserción.
    d. Si el usuario se ha movido con respecto a su última ubicación, se pinta en el mapa de calor de la interfaz un punto para indicar que en esa ubicación ha habido hallazgos.
    e. Repetir de ‘a’ hasta ‘d’, hasta que el usuario presione el botón de escaneo de nuevo.

Para que esta actividad se desempeñe correctamente es necesaria la intervención de las siguientes clases:

- **GeolocationGPS:** provee de una ubicación a la aplicación.
- **DatabaseHelper:** implementa los métodos CRUD que se emplearan para añadir los dispositivos y ubicaciones que corresponda.
- **MainActivity:** contiene la interfaz de escaneo y la funcionalidad de detección.

El código de las tres clases es demasiado extenso como para ponerlo en la memoria. Se puede encontrar en el Anexo físico entregado junto a la misma o en el repositorio de código indicado en la sección 2.1.

La Figura 8 presenta la interfaz principal de la aplicación móvil. El usuario, antes de presionar el botón de escaneo, se encuentra con dicha interfaz una vez el programa está cargado.

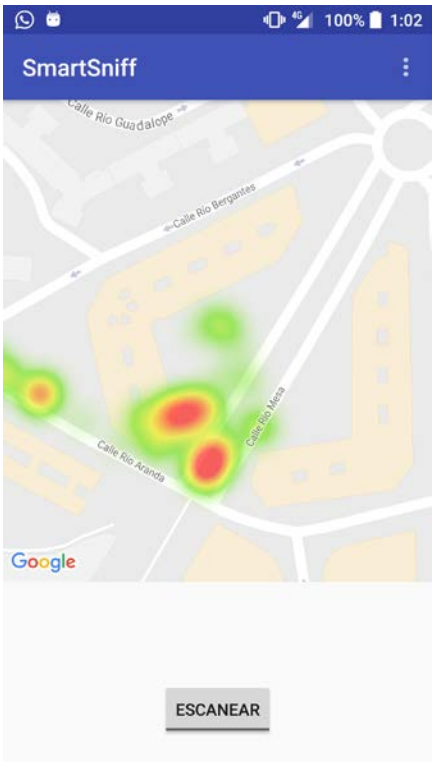

Figura 8 - Interfaz principal en modo *standby*

Cuando presiona el botón de escaneo, y tras verificar los permisos de ubicación y la interfaz Bluetooth, la interfaz muestra la información disponible de la sesión actual (fecha de inicio y número de hallazgos), como se muestra en la Figura 9. Es importante destacar que el número de hallazgos mostrado en esta interfaz es el número de dispositivos **nuevos** descubiertos.

Los dispositivos nuevos que se vayan descubriendo se asocian a la sesión en memoria de manera temporal, con el objetivo de no tener que recurrir a la base de datos a la hora de mostrar los resultados de la sesión terminada.

Si bien la obtención de la ubicación se hace de una forma muy habitual en este tipo de aplicaciones, la manera en la que se hacen las asociaciones es particular de esta aplicación.

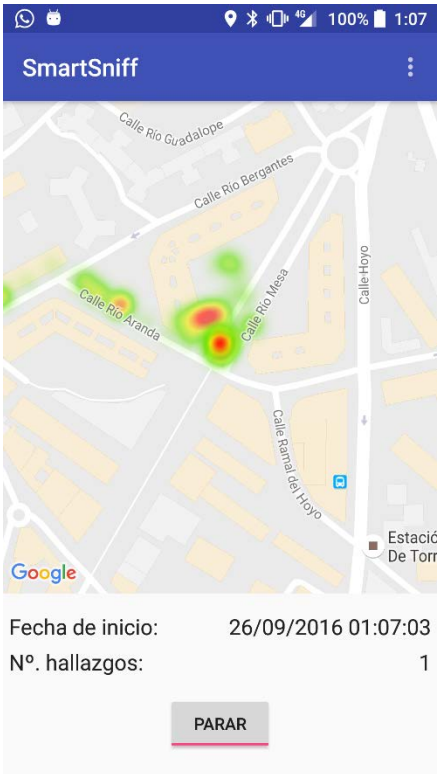

Figura 9 – Interfaz principal en modo escaneo

Únicamente se comentará la inserción de los dispositivos, puesto que la inserción de las localizaciones y las sesiones se hace de manera muy similar. A continuación se explica el método **addDevice()** de la clase **DatabaseHelper**. Dado que el código de dicho método es extenso no se ha expuesto aquí. No obstante, puede ser consultado en el Anexo físico de la memoria o en el repositorio de código de la aplicación móvil.

Debemos obtener una instancia en modo de escritura de la base de datos local y, seguidamente, preparar un cursor para almacenar el resultado de la consulta de inserción.

Preparamos un objeto ContentValues con los valores de las columnas de la tabla Device, obtenidos del dispositivo pasado como parámetro. Una vez hecho esto, creamos una variable en la que almacenaremos la ID que devuelva la consulta después de hacer la inserción. Esta ID es la ID de la fila de la tabla que correspondería al dispositivo añadido. En caso de error, esta consulta devolvería -1. No obstante, el error sería capturado por el bloque catch, y se procedería a su manejo.

En caso de que la consulta de inserción se realice con éxito y se devuelva la ID, se almacena en una variable de clase llamada **deviceId**. En el caso de que la consulta no pueda realizarse debido a que el dispositivo ya esté almacenado en la tabla, se obtiene la ID del mismo en la tabla mediante una búsqueda, utilizando para ello su dirección física ya que dos dispositivos distintos no pueden tener la misma dirección física. Esto se hace para garantizar que, tanto si se ha hecho bien la consulta como si no debido a que el dispositivo ya existe en la base de datos, se tenga la ID del dispositivo en la variable **deviceId** al finalizar el método.

A lo largo del proceso de trabajo se ha puesto un claro énfasis en llevar a cabo prácticas de programación defensiva. La descrita en el párrafo anterior es una de las muchas que hay distribuidas por todo el proyecto y, gracias a ellas, la aplicación puede mantener su funcionamiento de forma adecuada a pesar de que ocurran situaciones de error que se puedan controlar.

Esto mismo que se ha hecho con el dispositivo se realiza con la sesión y la ubicación en las variables **locationId** y **sessionId**, respectivamente.

Una vez se tienen las tres IDs, se llama al método **addAssociation()** de la misma clase. Nótese que la asociación se hace con las IDs obtenidas de haber realizado anteriormente las inserciones de la sesión, ubicación y dispositivo que intervienen en la asociación.

Ahora se procede a explicar el proceso de detección a nivel de código. Comenzaremos con los procesos de preparación para el escaneo, que tienen lugar cuando el usuario presiona el botón de escanear.

Primero se activa un *flag* que inhabilita el menú de opciones para evitar que el usuario pueda modificarlas mientras tiene lugar un escaneo.

Inmediatamente después se activa la interfaz de escaneo, y se procede a comprobar si la interfaz Bluetooth está activada, pidiéndole acto seguido al usuario que la active en caso contrario. En la llamada siguiente a **geoGPS.connect()**, la API de Google Play Services comprueba si los servicios de ubicación del teléfono están disponibles, y también se pide al usuario que los active si no lo están.

A continuación se obtiene la fecha actual del dispositivo y se almacena como fecha de inicio de la sesión. Se crea la sesión con dicha fecha y se añade la sesión a la base de datos para tener preparada la ID de dicha sesión en la tabla para cuando llegue el momento de hacer asociaciones. Nótese que únicamente se añade la fecha de inicio, la fecha de finalización se añade con una actualización posterior.

Para poder mantener un registro de los hallazgos de la sesión se crea la variable auxiliar **lastSession**, que contendrá a la sesión. Además, se crea la lista **lastSessionDevices**, que contendrá los resultados de la sesión para mostrarlos en la interfaz de resultados, más adelante. Es necesario destacar que esta lista tiene la propiedad de que es *thread-safe*, por lo que se accede a dicha lista de forma atómica.

Una vez hecho todo esto, se llama al procedimiento **beginScanningProcedure()**. El procedimiento de escaneo de dispositivos wifi se lleva a cabo en un *thread* que se ejecuta constantemente a intervalos de tiempo definidos por el usuario en la configuración. El tiempo por defecto son 3 segundos. El procedimiento de escaneo de dispositivos Bluetooth se realiza fuera del *thread*.

El BroadcastReceiver encargado de gestionar los resultados recibidos de los escaneos se registra en el contexto de la aplicación antes de comenzar los escaneos correspondientes. Antes de empezar el escaneo Bluetooth se comprueba de nuevo si la interfaz Bluetooth está activada para evitar empezarlo si no lo está, evitando así muchos errores.

Es importante destacar que el *thread* continúa ejecutándose mientras el botón de escaneo esté pulsado. En el momento en el que se vuelva a pulsar para desactivarlo, su ejecución termina.

El escaneo Bluetooth se trata de forma independiente puesto que es un proceso muy pesado tanto en coste de recursos como en tiempo de cómputo, por lo que se pide un escaneo cada vez que termina el anterior. Un escaneo Bluetooth tiene una duración estimada de unos 12 segundos. Está recomendado por Google como buena práctica pedir finalizar un escaneo Bluetooth aunque no se haya iniciado, para asegurar que no hay un gasto innecesario de recursos. Esto se hace en varios puntos de la aplicación, en especial durante el propio escaneo para verificar si el usuario ha desactivado la interfaz Bluetooth durante el mismo para evitar que se pidan escaneos hasta que se vuelva a activar de nuevo.

Todo el código que interviene en la detección de dispositivos puede encontrarse en el repositorio de la sección 2.1 o en el Anexo físico entregado junto a esta memoria.

5.3.3. Detener escaneo

Detener un escaneo en proceso pone en proceso una serie de eventos de especial importancia. Un resumen de dichos eventos es el siguiente:

1. Comienzan los procedimientos de parada de escaneo, que incluye la desvinculación del BroadcastReceiver encargado de procesar los resultados del escaneo, la actualización de la sesión y el refrescado del mapa de calor de la interfaz.
2. Se muestra al usuario una interfaz en la que se muestran los resultados.

Cabe destacar que, si bien ambos sucesos ocurren de forma secuencial, se muestran los resultados en una interfaz nueva mientras se actualiza el mapa de calor. Esto se consigue mediante el empleo de una tarea asíncrona (*AsyncTask)* encargada de procesar todos los puntos del mapa de calor mientras el usuario interacciona con la lista de resultados.

Las clases que intervienen en esta funcionalidad son **MapManager** y **MainActivity**. También se accede a la base de datos con los métodos de la clase **DatabaseHelper**. Para mostrar los resultados se recurre a las clases **ResultsActivity** y **ResultAdapter**.

Cuando el usuario presiona el botón de parar escaneo, ocurren varias cosas. En primer lugar se desvincula el BroadcastReceiver encargado de procesar los resultados de los escaneos. Si la interfaz Bluetooth está activada se pide cancelar cualquier escaneo Bluetooth en proceso, incluso si no lo hubiere. Seguidamente se desconecta el GPS y se actualiza la sesión en la base de datos con la fecha de finalización.

Inmediatamente después, se pide al **MapManager** que recargue el mapa de calor. El propósito de esto es mostrar los puntos del mapa como una mancha continua, ya que las inserciones se realizan punto a punto y la API es incapaz de procesar la mancha sobre la marcha.

A continuación se muestra una actividad con una interfaz que presenta al usuario los resultados de la sesión mientras que, paralelamente gracias al uso de una tarea asíncrona, se refresca el mapa de calor.

Figura 10 – Interfaz de resultados de sesión

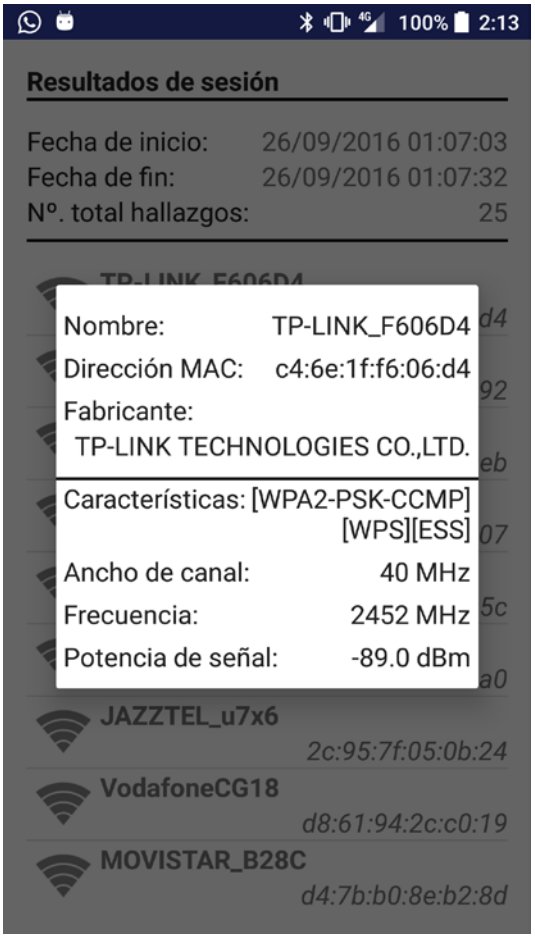

Figura 11 –Detalles de resultado

En la Figura 10 se pueden ver las fechas de inicio y fin de la sesión, con el número de hallazgos totales, tanto de dispositivos nuevos como ya encontrados, de la sesión. Debajo de dichos datos, se muestra una lista (vista de tipo *ListView*) con los resultados de la sesión. Se muestra un icono indicativo de si es un punto de acceso wifi o un dispositivo Bluetooth, el nombre (SSID) y la dirección física (BSSID).

Si el usuario pulsa sobre cualquier entrada de la lista, se muestra una interfaz en la que se exponen más detalles del dispositivo seleccionado. Dicha interfaz se expone en la Figura 11.

Es necesario mencionar que, para hacer la experiencia de navegar por la lista lo más cómoda posible para el usuario, se ha hecho uso del patrón de diseño **ViewHolder**. Este patrón consiste en que cada entrada de la lista almacena una referencia al elemento que representa, de manera que se tiene que representar la entrada solo una vez.

El código que implementa este patrón de diseño se encuentra en el método **getView()** de la clase **ResultAdapter**. Se muestra la parte más importante a continuación:

```
//First, check if the view is not null
if(convertView == null){
      //New view -> Inflate
      LayoutInflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
      v = layoutInflater.inflate(R.layout.result_row_layout, parent,
false);

      //Fill the layout with values
      TextView ssidTextView = (TextView)
v.findViewById(R.id.resultSsidTextView);
      TextView macTextView = (TextView)
v.findViewById(R.id.resultMacTextView);
      ImageView typeImageView = (ImageView)
v.findViewById(R.id.resultRowImage);

      holder.ssidTextView = ssidTextView;
      holder.macTextView = macTextView;
      holder.typeImageView = typeImageView;

      v.setTag(holder);
}
else{
      holder = (ResultHolder) v.getTag();
}
```

Código 3 – Patrón de diseño ViewHolder para la lista de resultados

Se puede apreciar que si la entrada de la lista (elemento **convertView**) no ha sido representada (es igual a null), se procede a "inflar" la interfaz dentro de la entrada y a asignar los datos a la misma. Acto seguido se guarda una referencia a los datos representados en un objeto de tipo **ResultHolder** con la llamada a **setTag()** pasando como parámetro el objeto. Si la entrada de la lista ha sido representada con anterioridad, no se tiene que inflar la interfaz de nuevo, puesto que se puede obtener el elemento representado (es decir, la entrada) rápidamente con una llamada a **getTag()**. Esto supone un ahorro en tiempo de cómputo considerable.

La llamada al refrescado del mapa de calor es una invocación a tres funciones simples implementadas en la API de mapas de calor de Google Maps, por lo que no se mostrarán en esta memoria. Puede consultarse el código si se desea en el Anexo físico.

5.3.4. Enviar datos

Para enviar los datos de la aplicación al servidor se hace uso de las librerías **Volley** y **Gson**. La primera es empleada para construir el objeto JSON y enviarlo en una petición HTTP, y la segunda se usa para serializar los objetos que representan a dispositivos, localizaciones, sesiones y asociaciones en el objeto JSON.

El código que construye el objeto JSON es muy extenso como para incluirlo en esta sección, pero puede consultarse en la clase **JSONGenerator**. Es muy mecánico y fácil de comprender.

A cada sesión se le añade como atributo la dirección física del adaptador de red del dispositivo que la envía. Esto es para diferenciarla del resto de sesiones que puedan haber empezado y terminado al mismo tiempo, si se diera a llegar tal situación.

Una vez se ha terminado de generar el objeto JSON, se manda por una petición HTTP de tipo POST a la API REST del servidor para que los datos sean añadidos a su base de datos. Antes de enviar datos se comprueba si hay datos que enviar almacenados. Si no fuera el caso, la aplicación muestra un mensaje de error al usuario.

5.3.5. Borrar datos

Para borrar los datos se emplea el método **deleteDatabase()** de la clase **DatabaseHelper**. El método consiste simplemente en ejecutar una sentencia SQL de tipo DELETE sobre cada una de las tablas, eliminando así toda la información de la base de datos local.

Primero, se comprueba que haya datos almacenados que borrar. Antes de proceder, se pide al usuario una confirmación mediante el mensaje de alerta mostrado en la Figura 12.

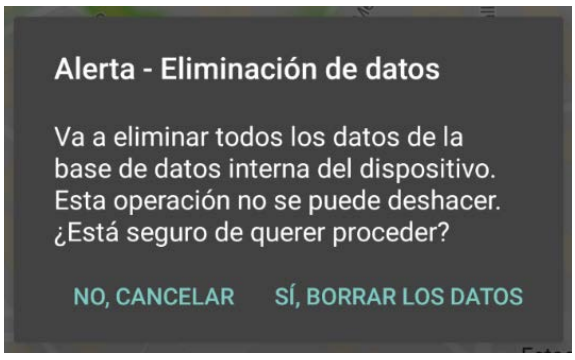

Figura 12 - Mensaje de confirmación de borrado de datos

5.3.6. Configuración

La funcionalidad de configuración es bastante sencilla. Consiste en una actividad que contiene los parámetros de la aplicación que el usuario puede ajustar. Se presenta la interfaz en la Figura 13.

Figura 13 – Interfaz de configuración

Se presentan dos opciones de configuración:

- **Modo de ahorro de batería:** con esta opción activada, la localización se obtiene de Internet en lugar del GPS. Se consigue por lo tanto un ahorro de batería significativo, perdiendo precisión en la ubicación obtenida.
- **Intervalo de escaneo:** tiempo en segundos que transcurre entre escaneos de dispositivos. Se ha marcado un intervalo de 2 a 10 segundos entre los cuales el usuario puede indicar el valor que desee.

Los parámetros que el usuario indique se guardan en las SharedPreferences de la aplicación Android, por lo que se puede acceder a ellos desde cualquier parte de la aplicación.

5.3.7. Aplicación web – Mapa de calor

Para elaborar el mapa de calor de la aplicación web se ha usado Leaflet, una de las librerías JavaScript más empleadas a la hora de desarrollar aplicaciones web con mapas interactivos. Para la funcionalidad de mapa de calor propiamente dicha se han empleado de forma conjunta los plugins **Leaflet.heat js** y **Heatmap.js.**

- **Leaflet.heat js:** este plugin es el que añade la funcionalidad de mapas de calor.
- **Heatmap.js:** este plugin, desarrollado por Patrick Wied y disponible de forma gratuita en su página web (https://www.patrick-wied.at/static/heatmapjs/), actúa por encima del plugin anterior proveyendo una interfaz para el desarrollador que simplifica mucho la tarea de crear mapas de calor. Posee un módulo de renderización capaz de renderizar más rápido los mapas de calor que Leaflet.heat js, además de ser más fácil de usar.

La obtención de los puntos a pintar en el mapa de calor se ha realizado de la siguiente manera:

- Mediante el empleo de las funciones **$(document).ready()** y **$.ajax()**, pedimos a uno de los métodos de nuestra API REST un objeto JSON que contendrá los datos a pintar en el mapa. **$(document).ready()** se encargará de que se pidan los puntos y se pinten en el mapa únicamente cuando el documento HTML esté cargado y listo, para evitar errores.
- El método que devuelve los datos para pintar es **GetHeatmapData()** de la clase **DbController**. Se ha creado una estructura de datos auxiliar, **HeatmapPoint**, que contiene los atributos necesarios para poder ser serializada en el JSON y leída en JavaScript de manera intuitiva:
    - **lat:** latitud del punto a representar.
    - **lng:** longitud del punto a representar.
    - **count:** valor de intensidad del punto a representar. En nuestro caso, contendrá el número de dispositivos detectados en esta posición.
- Una vez recibido y leído el JSON con los puntos, el mapa de calor es elaborado.

Se provee de una captura de ejemplo del mapa de calor en funcionamiento en la Figura 14.

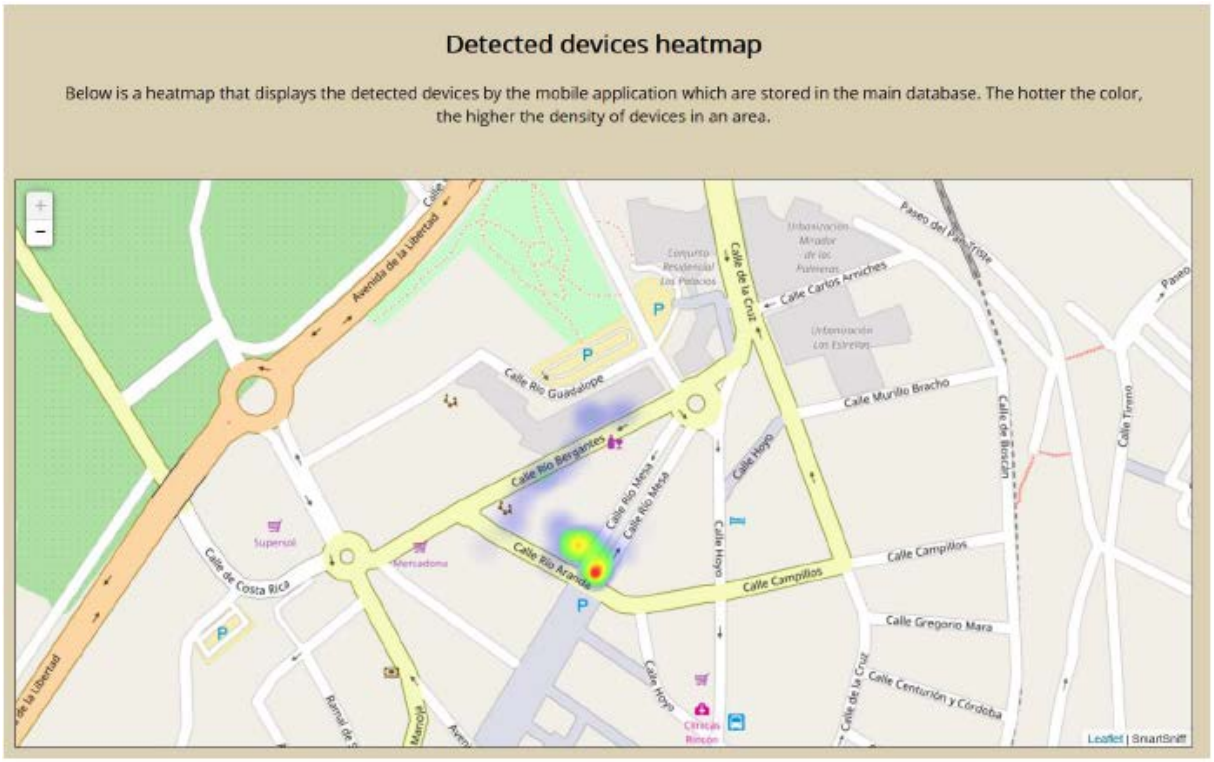

Figura 14 – Mapa de calor de la aplicación web

5.3.8. Aplicación web – Estadísticas

Para elaborar las estadísticas se ha procedido de manera similar al apartado anterior. Se ha optado por implementar cinco gráficas distintas:

- Dispositivos diarios detectados (diagrama de barras)
- Dispositivos encontrados clasificados por fabricante (gráfico circular)
- Ancho de canal más empleado (gráfico circular)
- Banda de frecuencia más empleada (gráfico circular)
- Dispositivos Bluetooth encontrados por tipo (gráfico circular)

Se ha programado un método en la API REST que devuelve un JSON con todas las estadísticas necesarias para elaborar los diagramas anteriores. Es muy extenso, por lo que no cabría en una página de esta memoria. No obstante, puede consultarse en la clase **DbController**. Es el método **GetStatisticsData()**.

De manera idéntica al mapa de calor, se han creado dos estructuras de datos auxiliares para poder elaborar el JSON de manera que sea fácil leerlo en JavaScript:

- **StatData:** contiene los campos **label** (nombre de la característica) y **value** (valor numérico).

- **StatArray:** no es más que un array de objetos StatData clasificados por diagrama a representar. Hay uno por cada gráfica, y cada uno de ellos contiene los datos necesarios para elaborar la estadística correspondiente.

El código de ambas estructuras de datos es el siguiente:

```
public struct StatData
{
	public string label { get; set; }
	public string value { get; set; }

	public StatData(String dataLabel, int dataValue)
	{
		label = dataLabel;
		value = dataValue.ToString();
	}
}

public struct StatArray
{
	public StatData[] statDataArray { get; set; }

	public StatArray(StatData[] dataArray)
	{
		statDataArray = dataArray;
	}
}
```

Código 4 – Estructuras de datos del controlador para las estadísticas web

La aplicación web permite al usuario filtrar los datos que se mostrarán en las gráficas por fecha. Se presentan a continuación capturas de dos de las cinco estadísticas empleadas, correspondientes a los dos tipos distintos de gráfica existentes en la aplicación (ver Figuras 15 y 16).

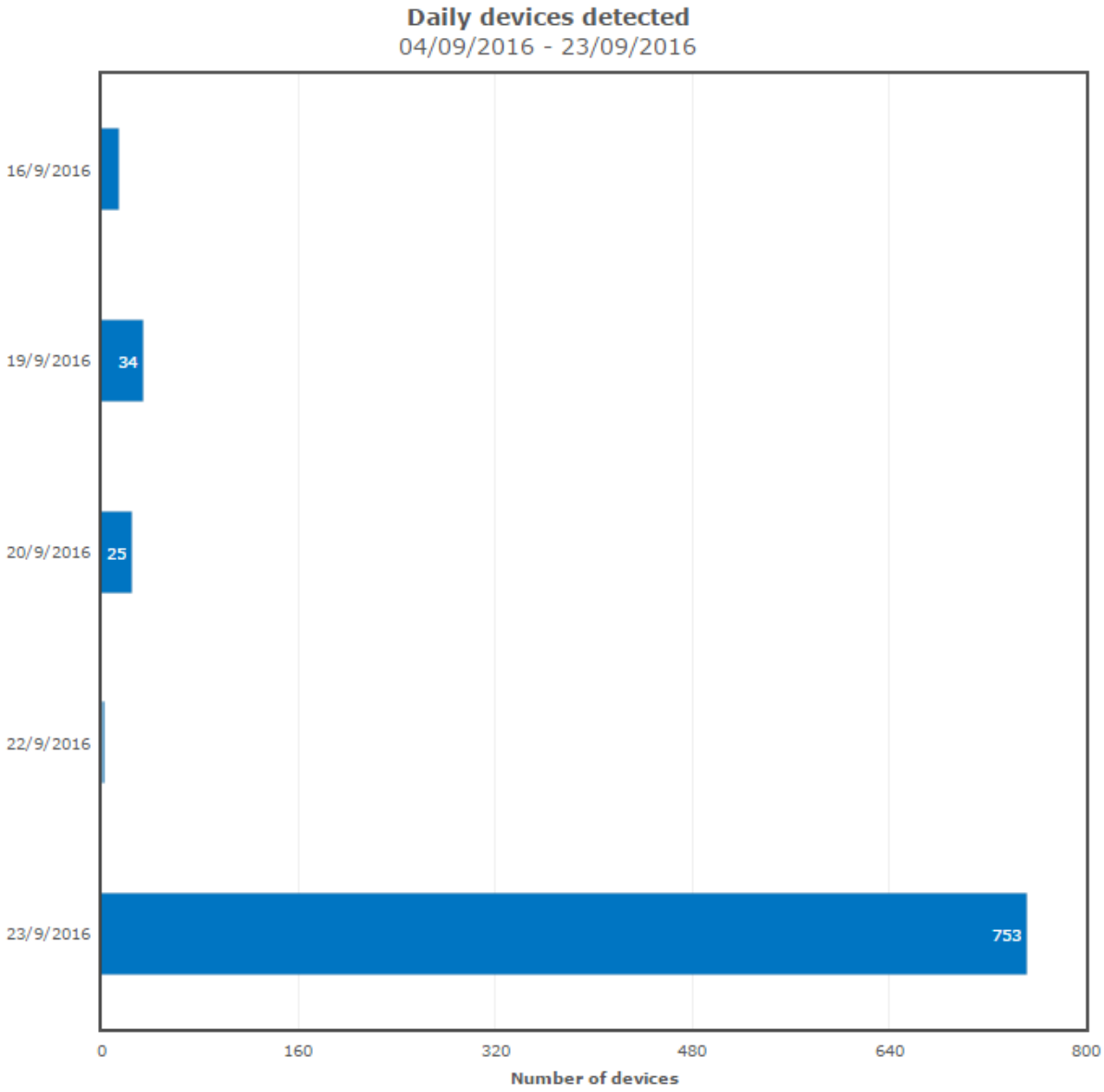

Figura 15 – Dispositivos diarios detectados (diagrama de barras)

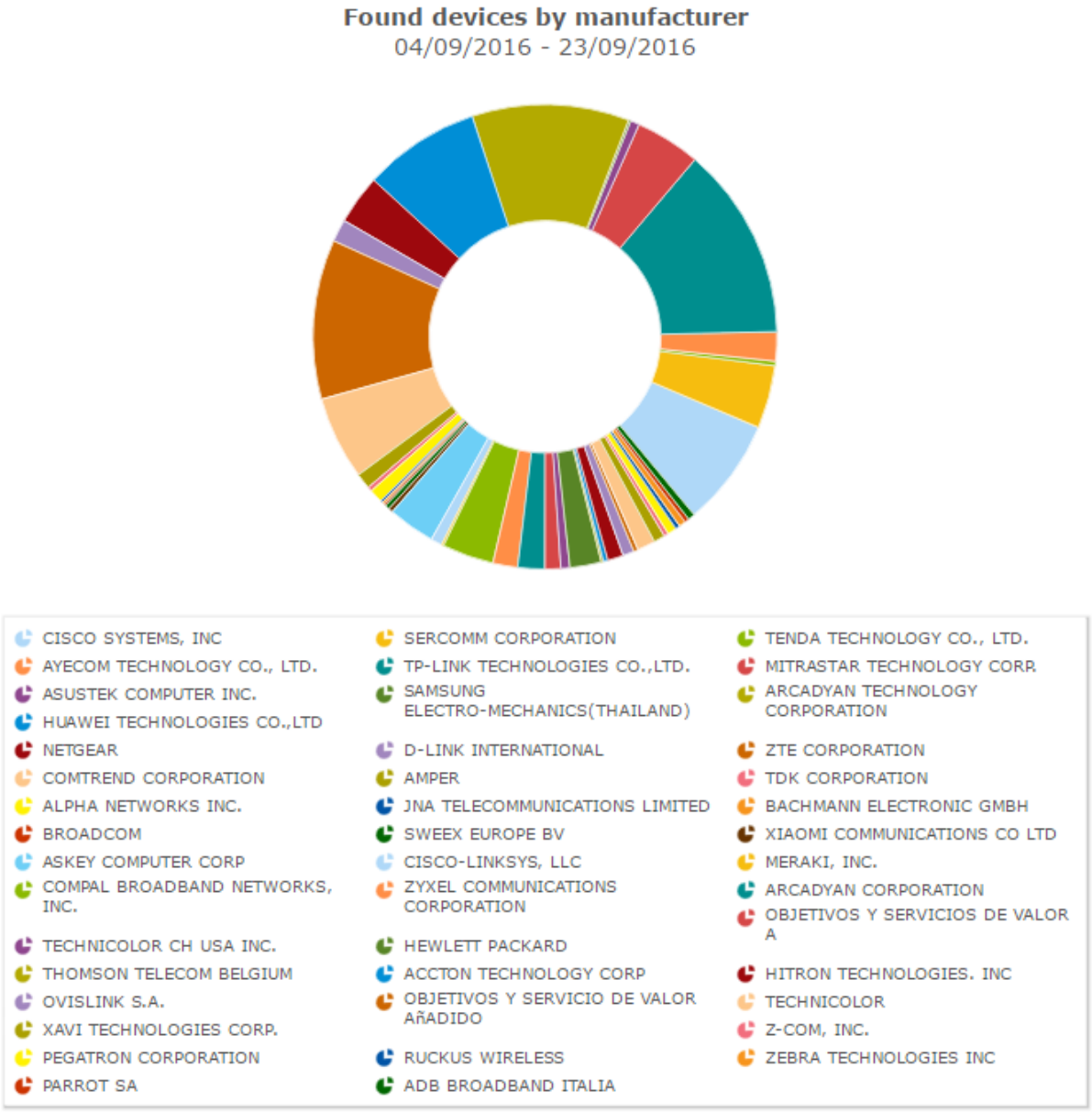

Figura 16 – Dispositivos detectados según fabricante (diagrama circular)

5.4. Pruebas

No se han podido realizar pruebas de código en la aplicación móvil, como se estimó inicialmente. La razón es que los emuladores Android no pueden simular ni localizaciones ni redes o dispositivos inalámbricos, y aquellos emuladores que aseguraban poder hacerlo presentaban situaciones de error, como por ejemplo cierres constantes del emulador. Asimismo, resultaba imposible cargar en el emulador las bibliotecas de Google Play empleadas para el desarrollo de la aplicación.

En su lugar, se han realizado pruebas en entornos urbanos reales para determinar el correcto funcionamiento de la aplicación y encontrar errores.

A continuación se presenta la última prueba de campo realizada. Dicha prueba de campo es de especial relevancia para el TFG, puesto que además de ser la de mayor envergadura permitió trabajar con un volumen de datos considerable y, además, descubrió un error crítico en la aplicación móvil.

Fecha de inicio	23/09/2016 18:18:35
Fecha de fin	23/09/2016 19:05:00
Número de dispositivos encontrados	973
Número de localizaciones registradas	513
Número de sesiones realizadas	3

Tabla 10 – Resumen de prueba de campo

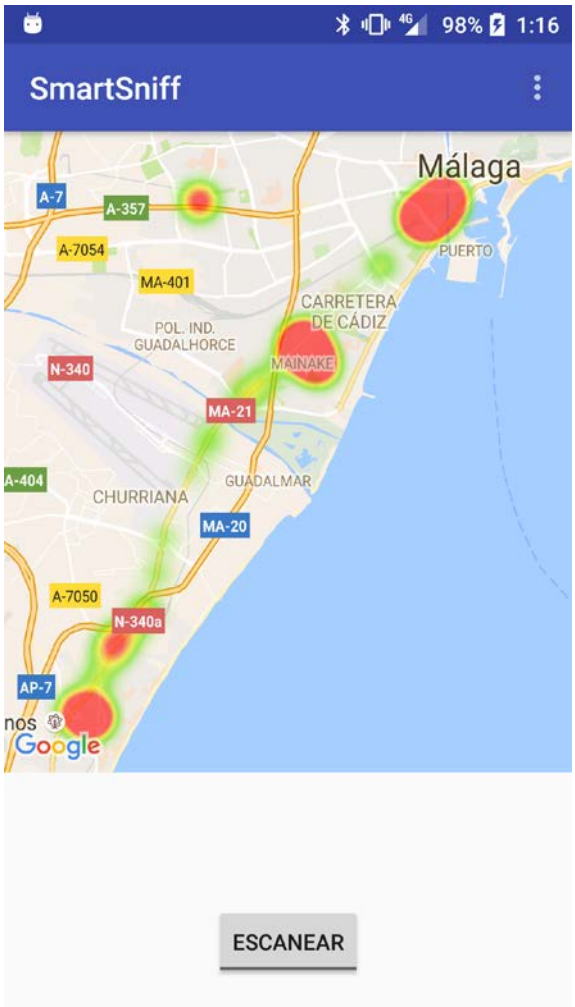

Figura 17 – Ruta seguida durante la prueba de campo

Primera sesión

Figura 18 – Pantalla de resultados de la primera sesión

La primera sesión fue un trayecto en coche. Como era de esperar, transcurrió sin ningún incidente y se recogió una amplia gama de resultados.

Una particularidad de esta sesión fue la aparición de muchos puntos de acceso wifi sin SSID asignado. Normalmente, cuando un punto de acceso no dispone de SSID suele tratarse de un repetidor o un dispositivo similar que no requiere de interacción alguna con los usuarios, por lo que es habitual que el SSID se omita.

No se encontraron muchos dispositivos Bluetooth. La inmensa mayoría de dispositivos (468) fueron puntos de acceso wifi.

Segunda sesión

Figura 19 – Pantalla de resultados de la segunda sesión

La segunda sesión se realizó en metro. La intención era comprobar el comportamiento de la aplicación móvil en situaciones de cobertura escasa o nula.

Los resultados obtenidos (ver Figura 19) fueron satisfactorios: la aplicación no tuvo ningún error que forzara su detención, y no registró los dispositivos encontrados sin haber una localización disponible, por lo que su comportamiento fue el esperado. En esta sesión, todos los dispositivos encontrados fueron puntos de acceso wifi.

De la tercera sesión no se dispone de pantalla resultados puesto que se encontró un error que hizo que la aplicación se detuviera forzosamente antes de mostrarlos. Dicho error era un fallo en el método que recargaba el mapa de calor de la interfaz tras un escaneo. Consistía en que una vez obtenidas las localizaciones de la tabla **Location**, al ejecutarse una función SQL COUNT para averiguar cuántos dispositivos tenía asociados cada localización (COUNT sobre **idLocation** en la tabla **AsocSessionDevice**) el programa tenía un error de desbordamiento de memoria.

La razón de dicho error era un cursor que no se cerraba, y que se encuentra dentro de un bucle que itera por todas las localizaciones encontradas. Se presenta el código a continuación (método **selectLocationsForHeatmap()** en **DatabaseHelper**).

```
public Map<Location, Integer> selectLocationsForHeatmap(){
      Map<Location, Integer> locationMap = new ArrayMap<>();

      SQLiteDatabase db = getReadableDatabase();
      Cursor cursor = null;
      db.beginTransaction();
      try{
            String selectQuery = "SELECT * FROM " + TABLE_LOCATIONS;
            cursor = db.rawQuery(selectQuery, null);

            if(cursor.moveToFirst()){
                  do {
                     String[] latlong =  cursor.getString(2).split(", ");
                     double latitude = Double.parseDouble(latlong[0]);
                     double longitude = Double.parseDouble(latlong[1]);
                     Location location = new Location(new
LatLng(latitude, longitude));

                     String countQuery = "SELECT count(*) FROM " +
TABLE_ASOCSESSIONSDEVICES + " WHERE " + KEY_ASSOCIATION_ID_LOCATION_FK +
" = " + cursor.getInt(0);
                     Cursor countCursor = db.rawQuery(countQuery, null);
                     if(countCursor.moveToFirst())
                        locationMap.put(location, countCursor.getInt(0));
                        countCursor.close();
                  }while(cursor.moveToNext());
            }
            db.setTransactionSuccessful();
      }
      . . .
      return locationMap;
}
```

Código 5 – Método selectLocationsForHeatmap()

El cursor que no se cerraba es **countCursor**, declarado para albergar el resultado de la consulta **countQuery**. La instrucción que arregló el error es **countCursor.close()**, situada justo debajo de la instrucción que coloca la ubicación junto al número de dispositivos detectados en ella en el Map que se devuelve como resultado de la función. Antes de hacer esto, la aplicación era capaz de cargar una cantidad limitada de puntos debido a la enorme cuantía de cursores que quedaban sin cerrar. Después del cambio, es capaz de procesar cualquier cantidad de ubicaciones.

Capítulo 6. Conclusiones y trabajo futuro

6.1. Conclusiones

Uno de los pilares fundamentales de las ciudades inteligentes es la interconexión entre dispositivos inalámbricos. Esto ha provocado un crecimiento desorbitado de redes wifi y dispositivos Bluetooth en entornos urbanos. Este TFG ha propuesto SmartSniff, un sistema basado en una aplicación móvil capaz de recolectar datos sobre los puntos de acceso wifi y dispositivos Bluetooth para elaborar estadísticas que pueden ser útiles tanto para usuarios como para instituciones.

La aplicación móvil permite recabar diversa información útil de los puntos de acceso wifi y dispositivos Bluetooth. Entre toda la información recabada, cabe destacar el nombre de dichos dispositivos, sus direcciones físicas y su fabricante, además de diversos parámetros como el tipo de dispositivo, el ancho de canal y la banda de frecuencia empleada para la emisión de datos.

Estos datos se envían a una aplicación web para elaborar estadísticas visuales, entre los cuales podemos encontrar un mapa de calor que refleja el número de dispositivos detectados en un área determinada o diagramas que representan los dispositivos encontrados categorizados por fabricante. A partir de estos datos podrían elaborarse otras estadísticas significativas que podrían ser utilizadas para la toma de decisiones supervisada. Un ejemplo de ello sería el uso de la marca de teléfono de una zona determinada para realizar o evaluar tareas de *marketing* adaptadas.

Este TFG ha supuesto una gran oportunidad para aprender muchas tecnologías nuevas, como Leaflet, .NET Core y Volley. El desafío fundamental del proyecto ha sido implementar un método de escaneo simultáneo de redes wifi y dispositivos Bluetooth por las particularidades del escaneo de estos últimos en dispositivos Android.

Desarrollar este sistema nos ha permitido comprobar que, tal y como se preveía, los entornos urbanos modernos están repletos de redes wifi. Abundan las redes privadas, aunque se puede apreciar que las redes públicas tienen cada vez más presencia en dichos entornos. Esto implica que cada vez más personas podrán beneficiarse de servicios que se ofrezcan a través de este tipo de redes.

También hemos comprobado, en claro contraste con lo comentado anteriormente, la escasez de dispositivos Bluetooth encontrados. Podría inferirse a partir de este resultado que los usuarios de dispositivos móviles a día de hoy no ven mucho sentido a llevar habilitada la interfaz Bluetooth de sus dispositivos móviles. Esto puede ser una consecuencia de la falta de servicios disponibles que empleen esta tecnología como base.

6.2. Líneas futuras

Del trabajo realizado en este TFG podrían partir las siguientes líneas de trabajo futuro. A continuación se presentan algunas de las más notables:

- **Muestra opcional del mapa de calor:** una mejora interesante de la aplicación móvil sería añadir como parámetro de configuración que se mostrase el mapa de calor de manera opcional. Al cargar el mapa cuando la aplicación se carga, hay un pico de descarga de datos que puede evitarse con esta opción. Si bien no es una cantidad preocupante para las tarifas de datos existentes hoy en día, es interesante el añadido de esta opción al menú de configuración.

- **Identificación de usuario**: añadir una identificación de resultados por perfil de usuario podría aumentar la experiencia de usuario con la aplicación y hacerla más positiva.

- **Exportación de datos:** poder exportar los datos almacenados en la base de datos de la aplicación móvil reduciría la cantidad de memoria que ocupa la aplicación.

- **Incorporación de herramientas de Inteligencia de Negocio (*Business Intelligence)*:** esta mejora nos permitiría generar conocimiento a partir de las grandes cantidades de información que se obtiene con este sistema.

Referencias

[1] Dameri, R. P., & Rosenthal-Sabroux, C. (2014). *Smart City*. Springer.

[2] García, F. M. (7 de Mayo de 2015). *Hipertextual.* Obtenido de https://hipertextual.com/2015/05/smart-cities-espana

[3] Página oficial SQLite: http://sqlite.org/docs.html

[4] Documentación Google Play Services API: https://developer.android.com/training/location/receive-location-updates.html

[5] Documentación Google Maps API - Heatmap: https://developers.google.com/maps/documentation/android-api/utility/heatmap#introduction

[6] Documentación Volley: https://developer.android.com/training/volley/index.html

[7] Guía de usuario Gson: https://github.com/google/gson/blob/master/UserGuide.md

[8] Página oficial PostgreSQL: https://www.postgresql.org/

[9] Shirhatti, S. *Publish to a Linux Production Environment.* Obtenido de ASP.NET Documentation: https://material.google.com/patterns/launch-screens.html#

[10] Página oficial .NET: https://dotnet.github.io/

[11] Página oficial Leaflet: http://leafletjs.com/

[12] Plugin Heatmap.js para Leaflet: https://www.patrick-wied.at/static/heatmapjs/plugin-leaflet-layer.html

[13] Página oficial FusionCharts: http://www.fusioncharts.com/

[14] Google. *Launch Screen - Patterns.* Obtenido de Material design guidelines: https://material.google.com/patterns/launch-screens.html

Apéndices

A. Manual de usuario

La aplicación está disponible en el Anexo físico entregado junto a esta memoria. También puede descargarse de forma directa leyendo el código QR de la Figura 20 o desde el enlace que se muestra justo a continuación.

https://drive.google.com/open?id=0B3RaqcBmjuboS2ZsZWNBcWxaSzg

Figura 20 – Código QR para descargar la aplicación Android

Una vez descargado el fichero APK que contiene la aplicación, podemos instalarlo accediendo a él en nuestro teléfono móvil. Para esto, se recomienda usar una app de gestión de ficheros que permita acceder a los ficheros descargados. Si tenemos Google Drive instalado en nuestro teléfono el proceso será más directo, ya que el fichero se abrirá automáticamente tras ser descargado.

Cuando abramos el APK, Android nos mostrará los permisos que requiere la aplicación y nos dará la opción de instalarla en nuestro teléfono. Cuando el sistema termine de instalar la aplicación, podremos ejecutarla.

La aplicación Android posee una interfaz muy simple e intuitiva. La interfaz principal se muestra en la Figura 21.

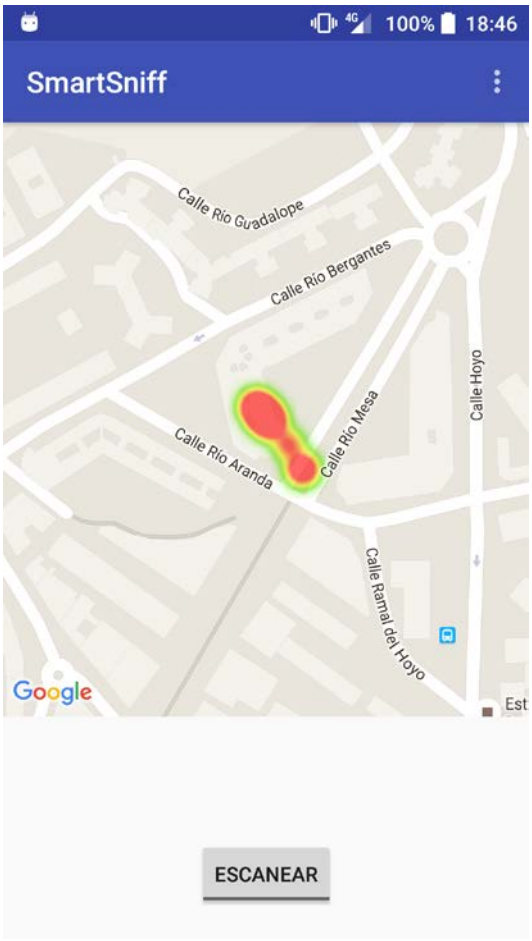

Figura 21 – Interfaz principal de la aplicación Android

El menú contextual de la barra de la aplicación (*appbar*) contiene el resto de funcionalidades de la aplicación. Pulsar sobre el botón del menú contextual hará que se despliegue el siguiente menú en la pantalla (ver Figura 22).

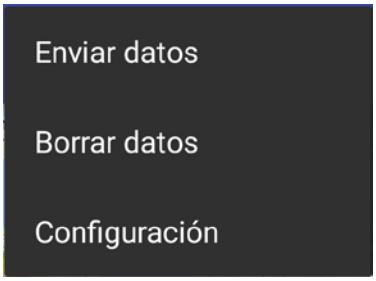

Figura 22 – Menú contextual de la appbar

Al presionar sobre el botón de enviar datos, la aplicación automáticamente recolectará todos los datos almacenados localmente y los enviará al servidor. Si no encuentra datos, mostrará un mensaje avisando de ello al usuario.

Al presionar sobre el botón de borrar datos, la aplicación primero comprueba si hay datos que borrar. Si no los hay, avisará de ello al usuario con un mensaje. En caso contrario, se mostrará el siguiente cuadro de diálogo (ver Figura 23).

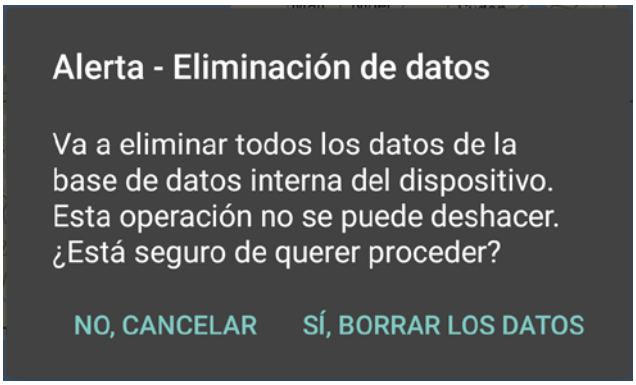

Figura 23 – Cuadro de diálogo para confirmar el borrado de datos

Si el usuario cancela la operación, la aplicación cancela el proceso de borrado. En caso contrario, la aplicación borrará todos los datos contenidos en la base de datos del dispositivo. Si se eliminan los datos, la aplicación avisará al usuario mostrando un mensaje de confirmación.

Al pulsar sobre el botón de configuración, la aplicación muestra una interfaz donde el usuario puede elegir si activar el modo de ahorro de energía y el tiempo que transcurre entre escaneos de dispositivos inalámbricos.

Al presionar sobre el botón de escanear, la aplicación comprueba que las interfaces de ubicación y Bluetooth estén activadas. En caso contrario pide permiso al usuario para activarlas.

Una vez hecho esto, se inicia el procedimiento de escaneo. Se detectan dispositivos Bluetooth a intervalos regulares de 12 segundos aproximadamente, mientras que el intervalo empleado para los dispositivos wifi puede indicarse en la interfaz de configuración.

El usuario puede finalizar el escaneo en cualquier momento presionando el botón de parar. Cuando esto ocurre, se muestra una interfaz con los resultados de la sesión. Presionando sobre cualquiera de los resultados mostrados se pueden obtener más detalles del mismo.

Se muestran a continuación las interfaces de escaneo y de muestra de resultados (ver Figuras 24 y 25).

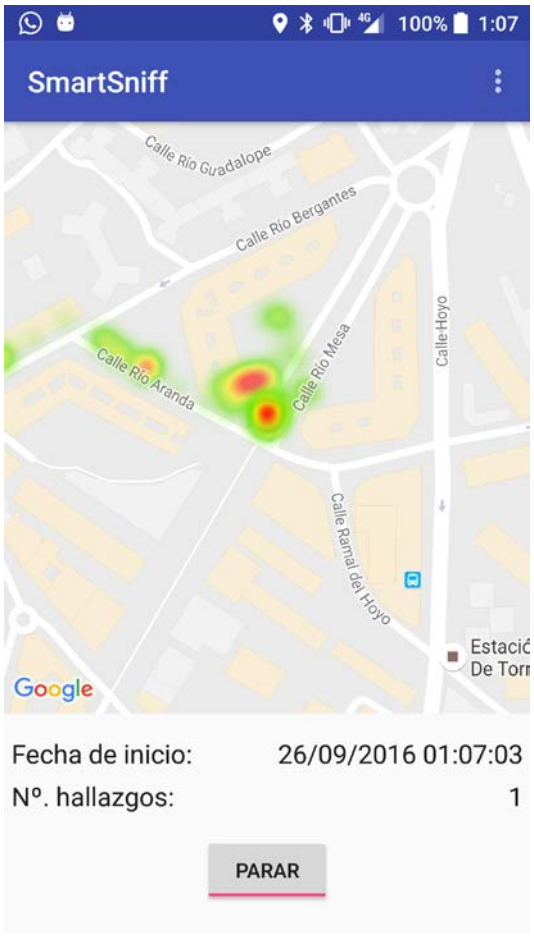

Figura 24 – Interfaz de escaneo

Figura 25 – Interfaz de muestra de resultados

B. Manual de instalación de la aplicación web

Este manual de instalación asume que el usuario posee un sistema Linux de 64 bits. La instalación que se documenta en este manual ha sido realizada sobre un sistema Ubuntu 16.04.1 LTS.

Paso 1 – Instalación de PostgreSQL y pgAdmin

Para instalar PostgreSQL en nuestro sistema, abrimos una instancia del terminal de Linux y ejecutamos la siguiente sentencia:

```
sudo apt-get install postgresql
```

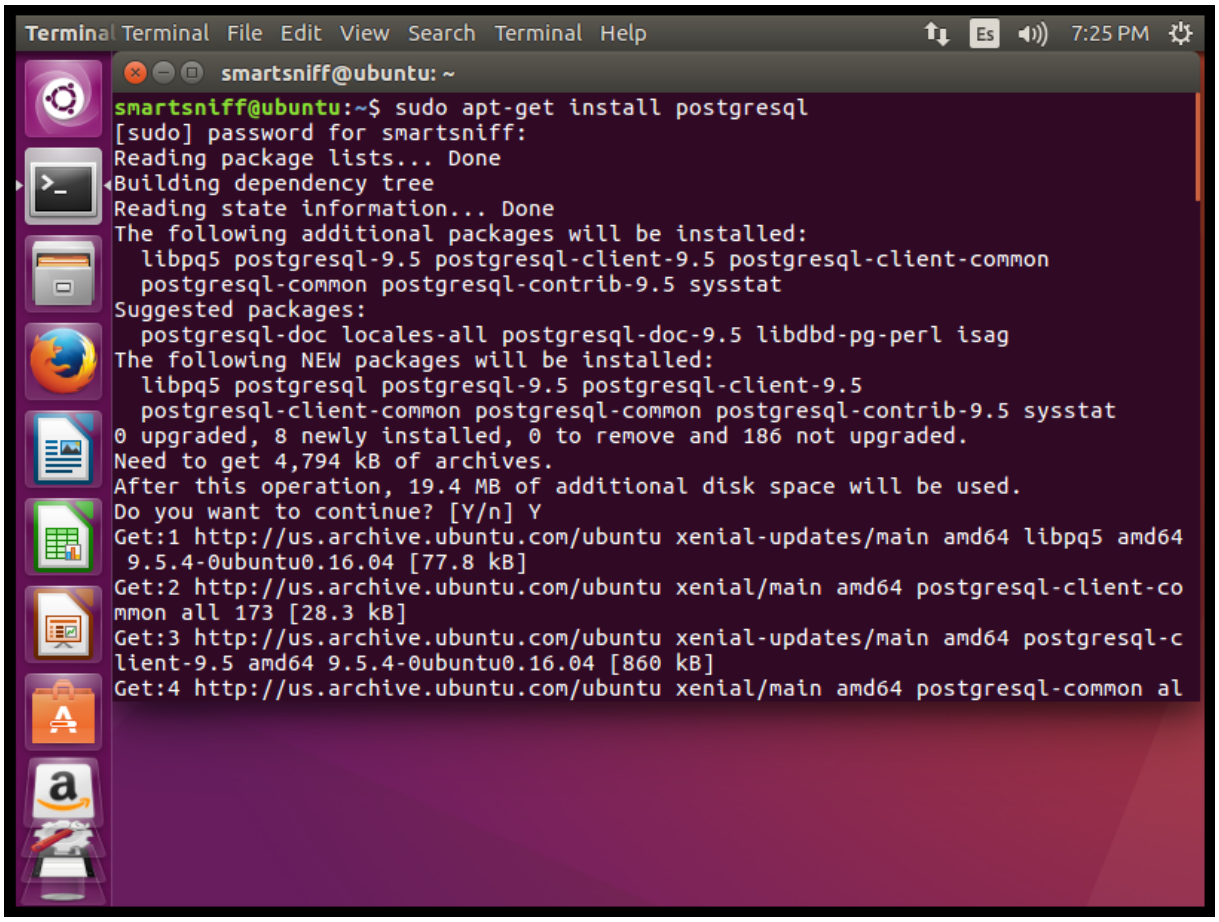

Figura 26 – Instalación de PostgreSQL con Terminal

Una vez hecho esto necesitaremos una interfaz gráfica para poder realizar el resto de nuestras tareas con la base de datos de forma más cómoda. Para ello, instalamos el software pgAdmin con la sentencia:

```
sudo apt-get install pgadmin3
```

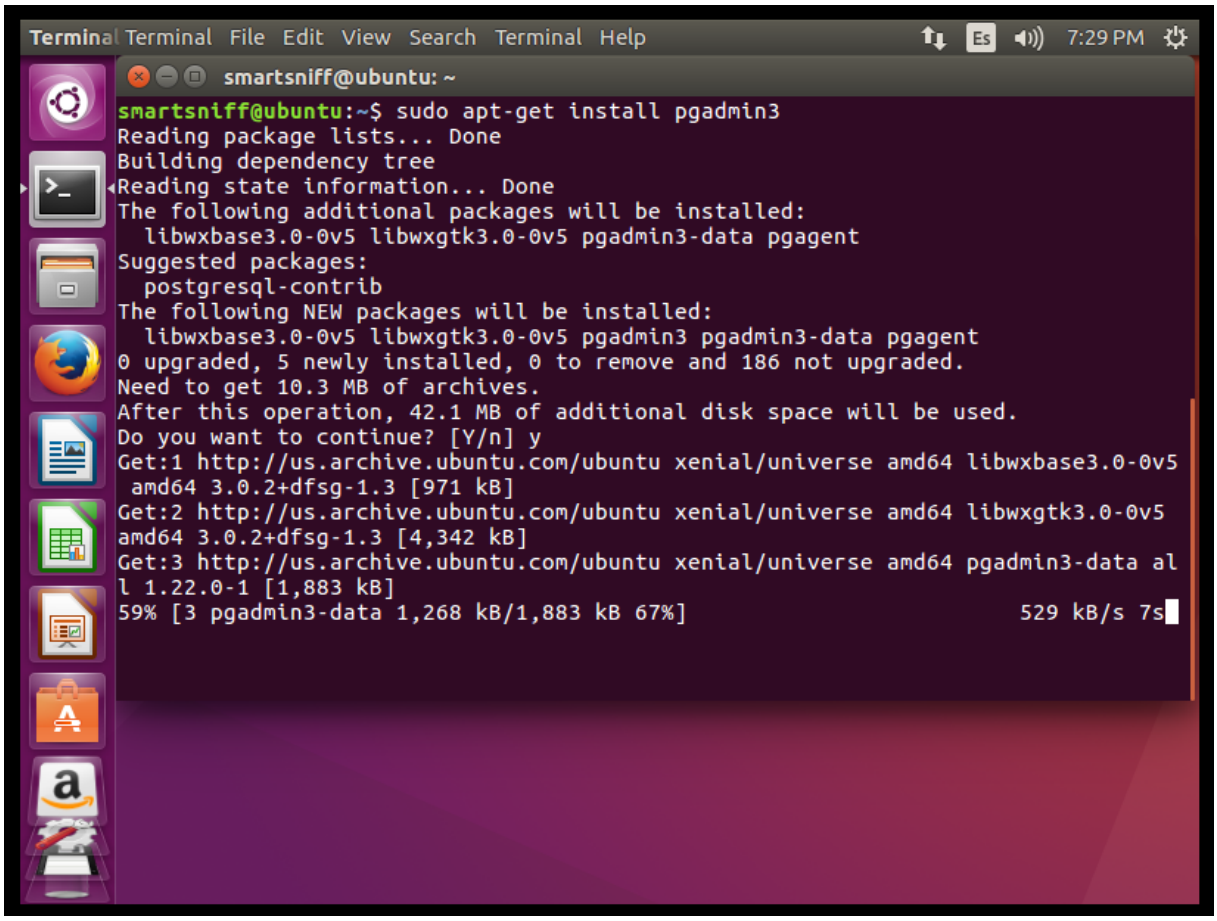

Figura 27 – Instalación de pgAdmin con Terminal

Paso 2 – Instalación de .NET Core

Antes de instalar el framework .NET Core es necesario actualizar las fuentes de paquetes del sistema Linux. Añadiremos la de .NET Core con las siguientes sentencias:

```
sudo sh -c 'echo "deb [arch=amd64] https://apt-mo.trafficmanager.net/repos/dotnet-release/ xenial main" > /etc/apt/sources.list.d/dotnetdev.list'

sudo apt-key adv –keyserver apt-mo.trafficmanager.net –recv-keys 417A0893

sudo apt-get update
```

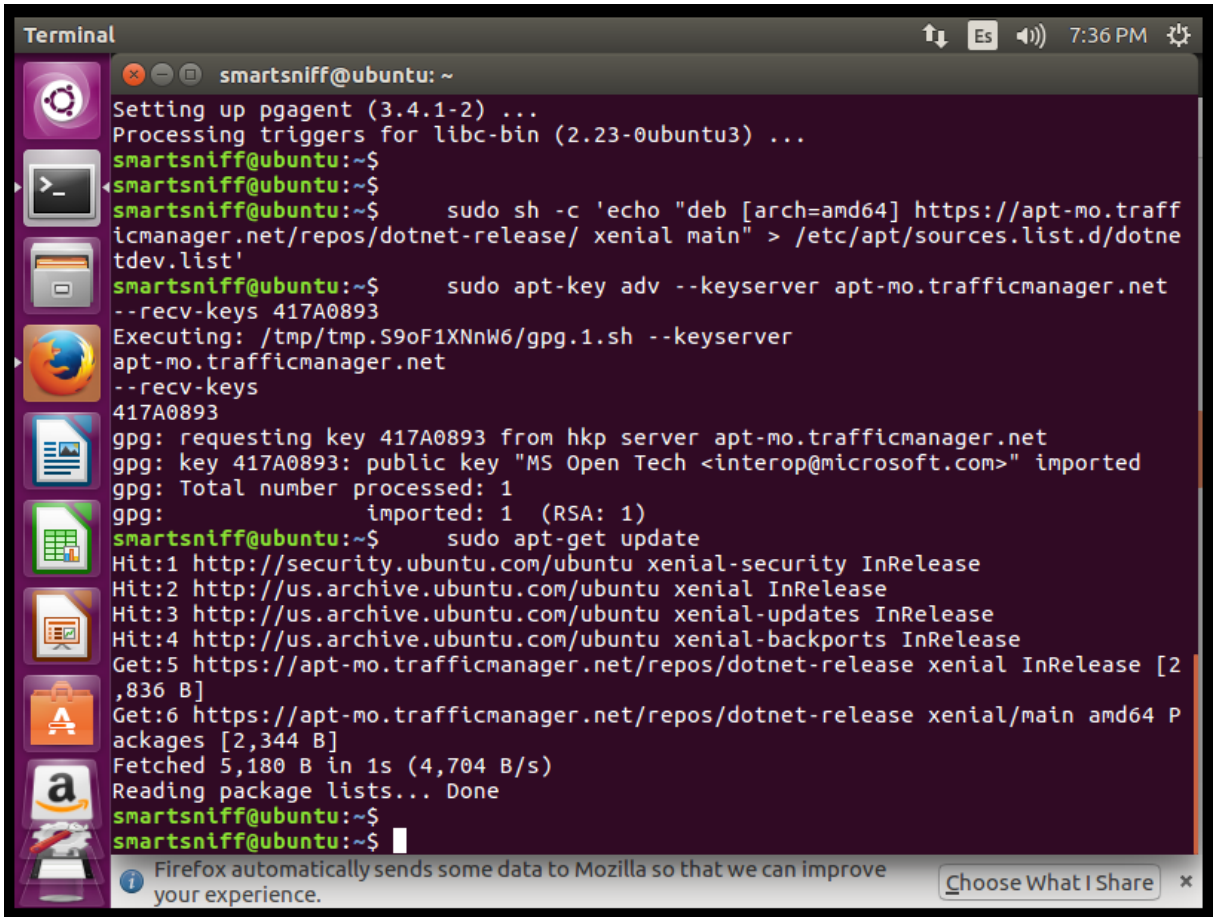

Figura 28 – Actualización de fuentes de paquetes de Linux con Terminal

Para finalizar este paso, instalaremos el propio framework con la siguiente sentencia:

```
sudo apt-get install dotnet-dev-1.0.0-preview2-003131
```

Es necesario mencionar que antes de ejecutar esta sentencia debemos eliminar cualquier versión previa de .NET Core que tengamos instalada en nuestro sistema. Para ello, podemos utilizar el script disponible en la propia sección de instalación de .NET Core para Linux, disponible en https://www.microsoft.com/net/core#ubuntu.

Una vez instalemos .NET Core y el sistema nos muestre la confirmación habitual de que la instalación se ha realizado correctamente, procederemos a configurar el servidor *proxy* inverso Nginx.

Paso 3 – Nginx

Instalaremos Nginx en el sistema ejecutando la sentencia

sudo apt-get install nginx

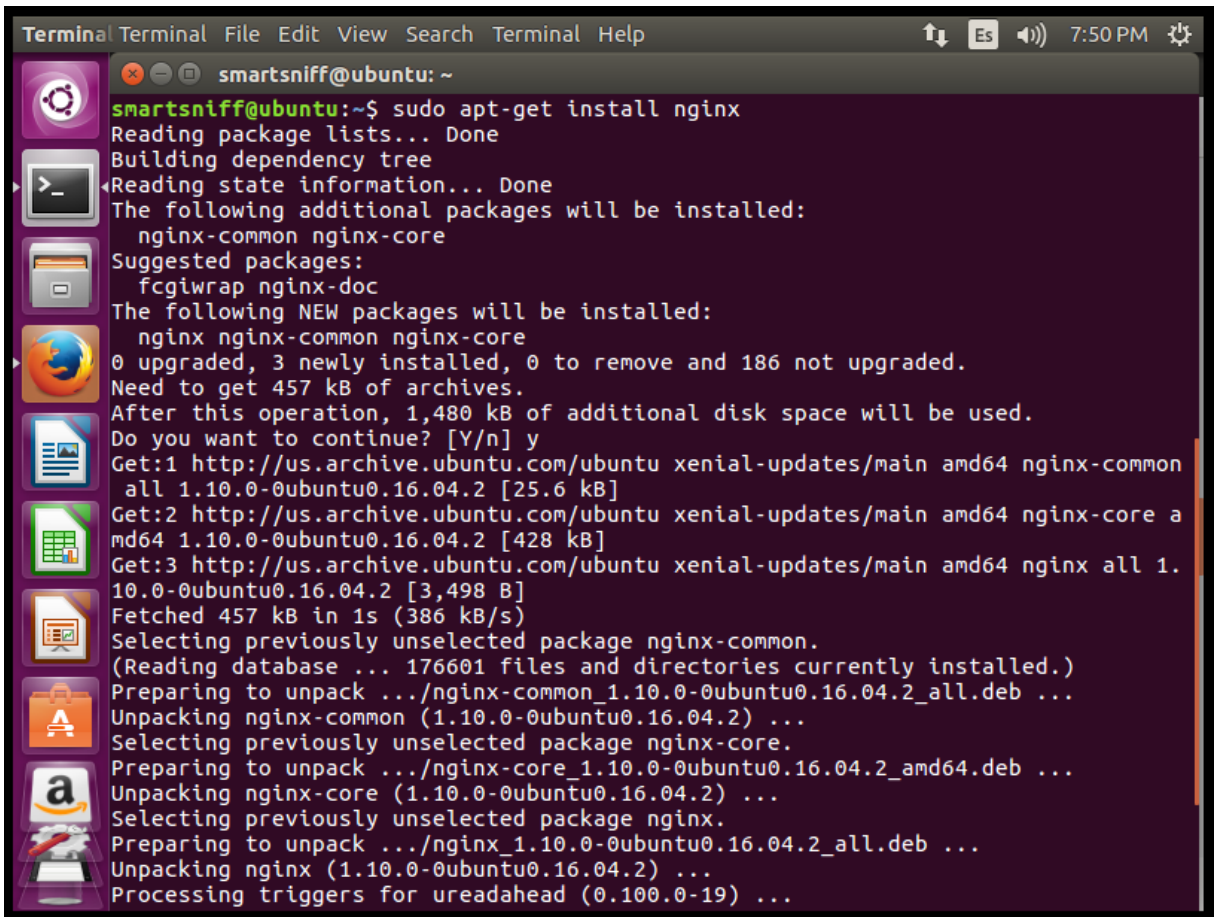

Figura 29 – Instalación de Nginx con Terminal

Nginx queda completamente instalado. Ahora necesitamos configurarlo adecuadamente. Lo primero que debemos hacer es eliminar el servidor de escucha por defecto de Nginx. Los comentarios se escriben con un símbolo # delante, así que deberemos abrir el fichero con ruta */etc/nginx/sites-available/default* con nuestro editor de texto favorito y comentar todas las líneas a excepción de aquellas que se encuentren dentro del ámbito *server*.

Puede abrirse el fichero directamente desde el terminal con la siguiente sentencia:

sudo nano /etc/nginx/sites-available/default

A continuación debemos configurar nuestro servidor modificando el fichero de configuración de Nginx, disponible en la ruta */etc/nginx/nginx.conf*. Eliminaremos todo lo que se encuentra dentro del ámbito http y escribiremos lo siguiente:

```
proxy_redirect off;
proxy_set_header Host $host;
proxy_set_header X-Real_IP $remote_addr;
proxy_set_header X-FOrwarded-For $proxy_add_x_forwarded_for;

client_max_body_size 10m;
client_body_buffer_size 128k;
proxy_connect_timeout 90;
proxy_send_timeout 90;
proxy_read_timeout 90;
proxy_buffers 32 4k;

limit_req_zone $binary_remote_addr zone=one:10m rate=5r/s;
server_tokens off;
sendfile off;
keepalive_timeout 30;
client_body_timeout 10;
client_header_timeout 10;
send_timeout 10;
```

Figura 30 – Configuración de Nginx (fichero nginx.conf)

Para finalizar, dado que nuestro servidor está escuchando en el puerto 5000, lo indicamos en el fichero *default.* Abrimos dicho fichero y escribimos lo que se muestra en la siguiente captura:

```
server {
        listen 80;
        server_name smartsnif.*;
        location / {
                proxy_pass http://localhost:5000;
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection keep-alive;
                proxy_set_header Host $host;
                proxy_cache_bypass $http_upgrade;
        }
}
```

Figura 31 – Configuración de Nginx (fichero *default*)

Tras esto, deberemos indicarle a Nginx que la configuración ha cambiado con el comando `sudo nginx -s reload`. Si lo hemos hecho todo bien, no debería haber errores.

Paso 4 – Puesta a punto de la base de datos

En este paso importaremos el esquema de la base de datos y configuraremos el usuario por defecto de PostgreSQL. En primer lugar, haremos lo segundo tal y como se muestra en la siguiente captura.

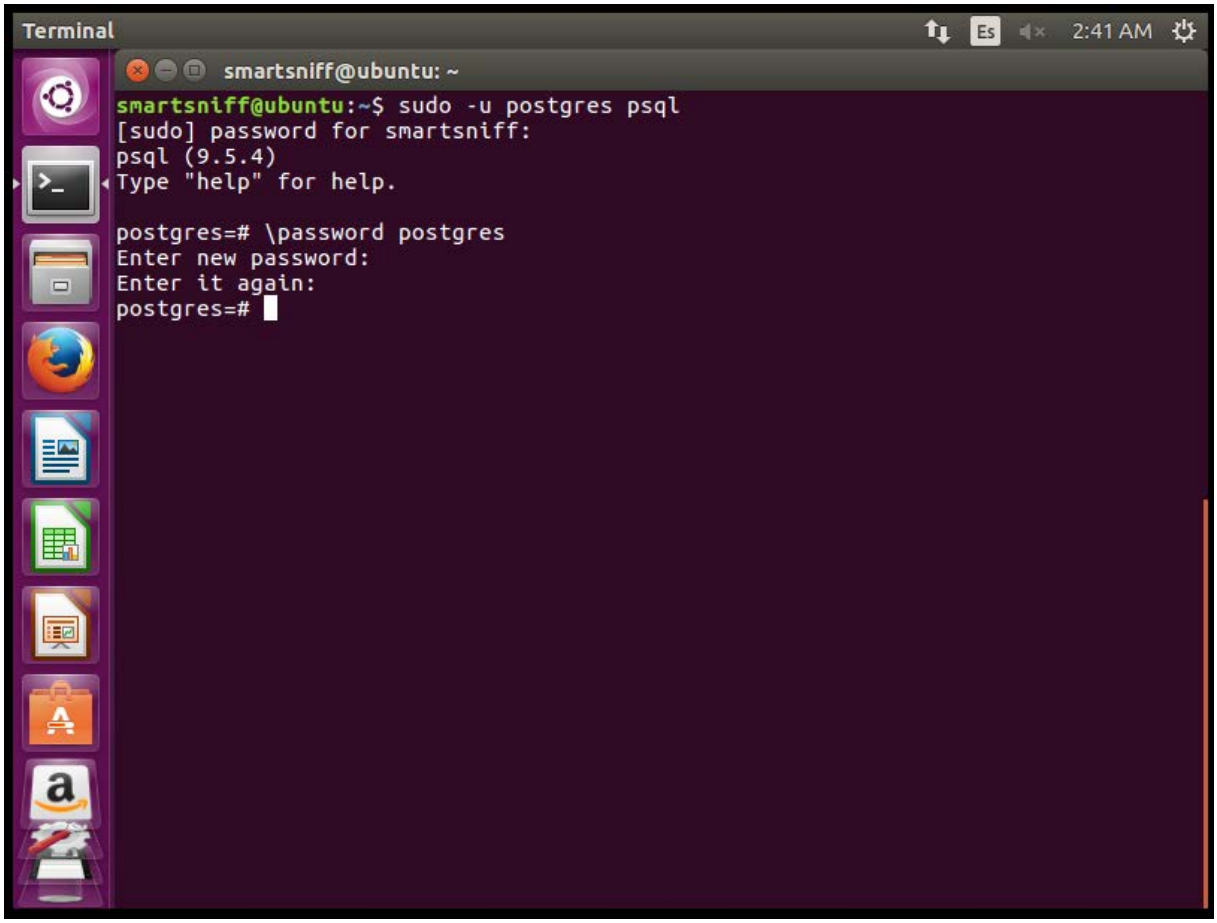

Figura 32 – Configuración del usuario postgres con Terminal

Inmediatamente después abriremos pgAdmin y crearemos una conexión nueva a la base de datos. Será en esta conexión donde importemos el esquema. La conexión se configura con el usuario postgres, tal y como se muestra en la Figura 33.

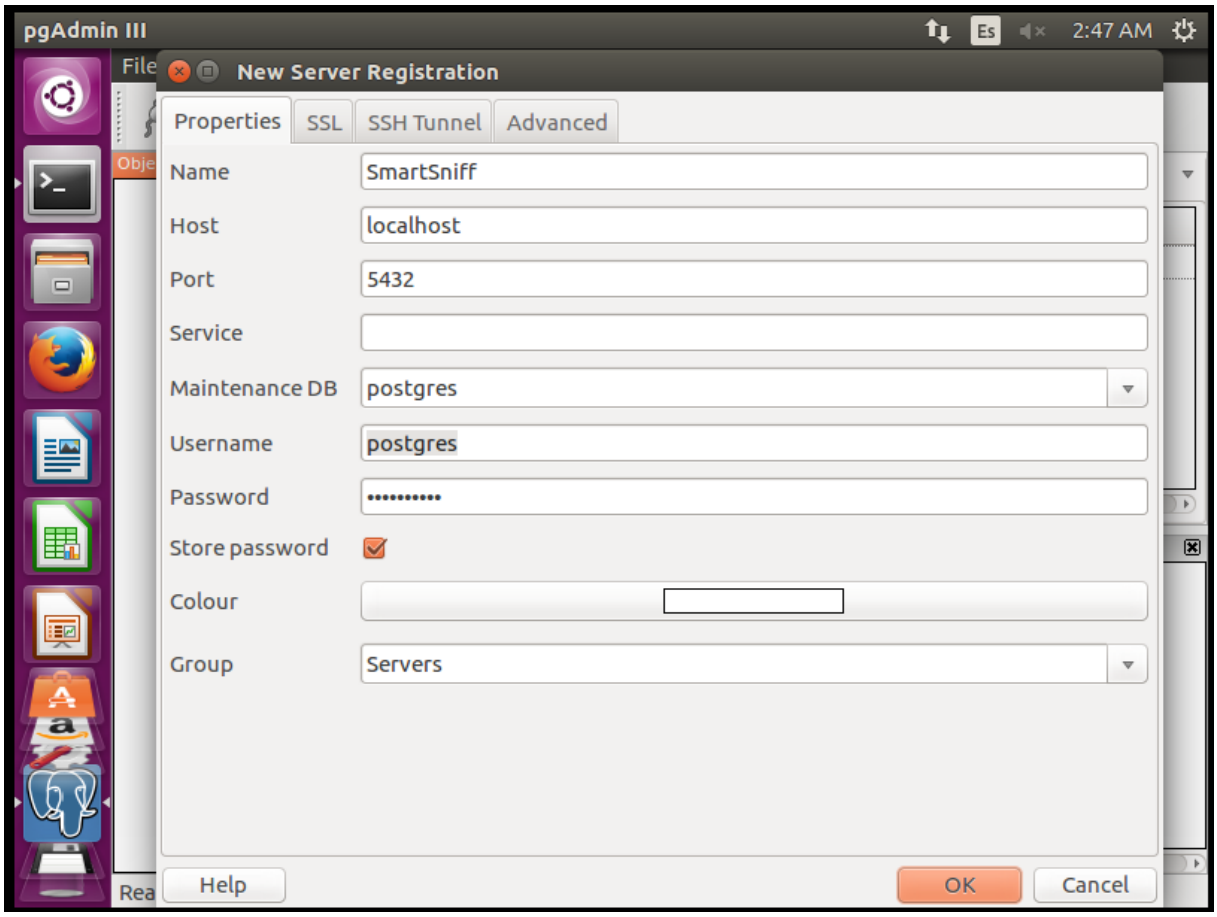

Figura 33 – Configuración de la conexión al servidor de bases de datos

Una vez hecho esto abrimos el desplegable de la conexión hasta encontrarnos con el de las bases de datos. Hacemos click derecho sobre el mismo y después seleccionamos la opción "*New Database*". La llamaremos "smartsniff-db" y asignamos el usuario *postgres* como *Owner.*

Finalmente hacemos click derecho en la base de datos recién creada y después en "*Restore*". Seleccionamos nuestro esquema de base de datos y hacemos click en "*Restore*". Al finalizar el proceso, la base de datos quedará totalmente funcional. El paso final sería importar la aplicación en .NET Core a nuestro sistema linux y, en la carpeta donde se encuentre el dll de la aplicación, escribir el siguiente comando en el terminal:

```
dotnet 'dll de la aplicación'
```

C. Glosario

Todos los términos empleados en este documento relacionados directamente con la Ingeniería del Software están definidos en el estándar **IEEE Std 610.12-1990 Standard Glossary of Software Engineering Terminology.**

HTML: *Hyper Text Markup Language.* Lenguaje declarativo para la elaboración de páginas web.

CSS: *Cascading Style Sheets.* Lenguaje para la definición y creación de la presentación visual de un documento HTML.

JS: abreviatura de JavaScript, lenguaje de programación interpretado empleado para elaborar scripts que se ejecutan en páginas web.

AJAX: *Asynchronous JavaScript and XML*. Técnica de desarrollo web que permite hacer peticiones asíncronas a una base de datos para actualizar una página web sin necesidad de recargarla.

HTTP: *HyperText Transfer Protocol*. Protocolo de comunicación que permite las transferencias de información por Internet.

JSON: *JavaScript Object Notation*. Formato de texto ligero para el intercambio de datos ampliamente usado en la actualidad.

API: *Application Programming Interface*. Conjunto de funciones y métodos que son ofrecidos por bibliotecas software para ser empleados como capa de abstracción.

REST: *Representational State Transfer*. Interfaz entre sistemas que usa HTTP de forma directa para obtener datos o indicar la ejecución de operaciones sobre datos. Sus principales características son un rendimiento rápido, una robustez elevada y su escalabilidad.

CRUD: *Create, Read, Update, Delete*. Funcionalidades básicas en bases de datos.

APK: *Android Application Package*. Paquete contenedor de componentes para el sistema operativo Android. Usado para distribuir e instalar aplicaciones.

Índice de figuras

Índice de tablas